ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE HONEYNET MEDIANTE UNA RED DEFINIDA POR SOFTWARE (SDN)

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

ANDRÉS MICHAEL MANZANO BARRIONUEVO

andres_manzano017@hotmail.com

JOSÉ LUIS NARANJO VILLOTA

joseluisnaranjo@hotmail.es

DIRECTOR: ING. DAVID MEJÍA, MSc.

david.mejia@epn.edu.ec

Quito, Septiembre 2015

DECLARACIÓN

Nosotros, Andrés Michael Manzano Barrionuevo, José Luis Naranjo Villota, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Andrés Michael Manzano Barrionuevo

José Luis Naranjo Villota

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Andrés Michael Manzano Barrionuevo y José Luis Naranjo Villota, bajo mi supervisión.

_____

Ing. David Mejía, MSc.

DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

Agradezco a cada una de las personas con las que me encontré en esta etapa de mi vida, y las cuales supieron aportar en mi formación personal y profesional, con las que día a día aprendí y compartí experiencias enriquecedoras, a los que puedo decirles amigos.

A mi familia que me supo apoyar y aconsejar en cada paso que daba y cuando se me presentaba una dificultad, supieron como animarme y seguir a delante.

Un agradecimiento especial a mi compañero y al director de este proyecto.

Andrés Manzano

AGRADECIMIENTOS

Gracias a Dios, por su protección, por todas las bendiciones que me ha otorgado y por llevarme por un buen camino.

Agradezco el apoyo incondicional de mi familia, gracias por siempre confiar en mí, porque esa confianza me daba la fuerza necesaria para nunca rendirme.

A la Escuela Politécnica Nacional y a mis profesores, por haberme formado como profesional.

A todos los amigos que me tendieron una mano cuando los necesitaba, siempre estarán presentes en mi corazón y en mi mente.

Gracias a mi director y compañero de este proyecto, porque juntos hemos trabajado en armonía y realizamos un muy buen trabajo.

José Luis Naranjo

DEDICATORIA

Dedico a mi madre de la cual aprendí a ser la persona que soy, a mi padre que aun que no pudo estar presente supo darme ese empujoncito cuando lo necesitaba.

Andrés Manzano

DEDICATORIA

Este logro es para mis padres Luis Aníbal y Emma Beatriz, los amo con mi vida.

José Luis Naranjo

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE SEGMENTOS DE CÓDIGO

RESUMEN

En este proyecto se presenta en un principio un breve resumen de la tecnología de Redes Definidas por Software (SDN), a continuación también se muestran las tecnologías y herramientas utilizadas para la implementación de una *honeynet*, y posteriormente se analizan ciertos ataques de seguridad informática en los que se presenta la forma de efectuarlos y el funcionamiento de los mismos.

Posteriormente, se muestra la implementación y desarrollo de una aplicación escrita en Python para el controlador denominado Pyretic, que haciendo uso de la tecnología de Redes Definidas por Software permite detectar seis diferentes tipos de ataques informáticos, tres de los cuales pertenecen al tipo DoS (*Denny of Service*) y los tres restantes al tipo suplantación de identidad (*Spoofing*).

La aplicación que corre sobre el controlador dispone de varios módulos, para la detección de los ataques, para determinar que paquetes se debe permitir dejar pasar de forma transparente hacia la LAN y cuáles deben ser desviados hacia la *honeynet*. Es decir, la aplicación desarrollada permite desviar el ataque. No se intenta contrarrestarlo o bloquearlo, esto con la finalidad de que el ataque continúe sin que el atacante perciba el cambio y de esta manera se logre recolectar la mayor cantidad de información posible respecto a la forma de operar del mismo y de los métodos o herramientas que este utilice.

Adicionalmente se presentan los algoritmos utilizados mediante diagramas de flujo y finalmente se muestra el código implementado y los datos obtenidos al realizar las pruebas de funcionamiento respectivas.

PRESENTACIÓN

Las redes de datos se han convertido en los últimos años en una necesidad fundamental que abarca muchos sectores importantes, como por ejemplo en los estudios, en el trabajo, en el comercio, en la comunicación, en los juegos en línea, entre otros.

Estos sectores, demandan continuamente nuevos requerimientos de las redes de datos y a medida que se incorporan nuevas funcionalidades, estas crecían con rapidez.

Debido a que las redes tradicionales, no fueron diseñadas para satisfacer muchas de las necesidades actuales, se empezaron a hacer modificaciones que permitieron dar soluciones temporales, sin embargo, el incremento interminable de nuevas necesidades están llevando a las redes actuales a su límite.

Es por tal motivo que se ha buscado una solución que permita hacer a las redes de datos redes más programables, agiles y flexibles y por supuesto que permitan innovar.

Las redes SDN brindan estos beneficios y por tal razón desde los inicios de internet, se empezó a incursionar en este campo y finalmente en el año 2010 ya se contaba con una interfaz de programación de aplicaciones para OpenFlow, el protocolo desarrollado para la comunicación del plano de control con el plano de datos.

Ante esta situación, en la que muchos investigadores están actualmente trabajando, se presenta en este proyecto una solución enfocada en la seguridad informática haciendo uso de esta nueva arquitectura de red de datos que por muchos es ya considerada como la tecnología de próxima generación.

El prototipo de *honeynet* que se ha desarrollado permite detectar ciertos ataques informáticos y desviarlos hacia una *honeynet* para posteriormente poder monitorearlos y analizarlos con mayor detenimiento.

CAPÍTULO I

1. MARCO TEÓRICO

En este capítulo se presenta un breve compendio sobre la tecnología SDN[1]; el mismo que incluirá la definición, las características, el funcionamiento y una comparación sobre dicha tecnología en contraste con las redes tradicionales. Posteriormente se adjunta un conciso estudio de las *honeynets*, en el que se trata su definición, características, ventajas, desventajas, requerimientos de implementación, arquitectura, estado del arte de las herramientas de despliegue y finalmente un análisis comparativo entre ellas. Luego de ello se muestran ciertas herramientas que permiten realizar ataques informáticos y finalmente se hace un breve estudio de las técnicas de ataque de las que se pretende proteger en este proyecto.

1.1 REDES DEFINIDAS POR SOFTWARE (SDN)

Las arquitecturas de red tradicionales han experimentado en los últimos años, una tendencia muy significativa, y por lo tanto muy notoria a disminuir su capacidad de brindar soluciones adecuadas o eficientes a problemas tanto de programación como de administración de red y por ende a recuperación de fallos ante eventuales problemas, la tecnología SDN ha empezado a desarrollarse, con el fin de reemplazar dichas arquitecturas y brindar una solución que sea más dinámica, manejable, rentable y por supuesto adaptable a cualquier tipo de red, inclusive si se trata de manejar soluciones que incluyan equipos de diferentes fabricantes.

Con una red SDN, el administrador puede darle forma al tráfico desde una consola de control centralizado sin tener que ir configurando cada uno de los diferentes equipos de networking de los que se disponga. El administrador puede cambiar el comportamiento de los dispositivos de la red en cualquier momento asignando o quitando prioridad, o inclusive bloqueando tipos específicos de paquetes con un nivel de control detallado.

---

[1] SDN (Software Defined Networking) es un esquema de redes de computadores cuyo objetivo es facilitar la implementación de servicios en la red para lo cual separa el plano de control del plano de datos de los equipos.

1.1.1 DEFINICIÓN

Las Redes Definidas por Software son una nueva arquitectura de red que aborda de una manera diferente la creación de redes, para lo cual hace que el control se separe del hardware y se lo da a una aplicación de software que se llama controlador. Esta arquitectura de red permite entonces, separar el plano de control del plano de datos buscando conseguir redes que sean más programables y flexibles.

SDN al hacer esa separación elimina completamente la inteligencia del hardware de las redes tradicionales, y se la da a un controlador. El control es independiente de cada capa del modelo de referencia OSI[2], y gracias a esto es posible reducir el costo y la complejidad para tener un mejor rendimiento de la red.

En una red tradicional se tratan los diferentes paquetes de acuerdo a las reglas especificadas en el firmware del dispositivo, y eso únicamente permite tratar a todos los paquetes de la misma forma sin poder dar un trato prioritario de acuerdo al tipo de tráfico o a un valor de prioridad. La tecnología SDN da solución a este problema de una forma más económica de la que lo hizo la introducción de circuitos integrados ASIC[3].

Según la "*Open Networking Foundation*", las redes definidas por software son una arquitectura emergente, dinámica, manejable, rentable, y adaptable, lo que es ideal para el gran ancho de banda y la naturaleza dinámica de las aplicaciones de hoy. Esta arquitectura desacopla el control de la red y las funciones de reenvío, permitiendo así que el control de la red sea directamente programable y que la infraestructura subyacente sea abstracta para las aplicaciones y servicios de red [1].

Con esta tecnología se virtualiza la red independizándola de la infraestructura física que se tenga.

---

[2] OSI (*Open System Interconnection*) es el modelo de red descriptivo creado en 1980 por la Organización Internacional de Estandarización.

[3] ASIC (*Application Specific Integrated Circuit*) circuito integrado hecho a medida para un uso específico.

1.1.2 CARACTERÍSTICAS

Las Redes Definidas por Software ayudan a gestionar la naturaleza dinámica de las aplicaciones de hoy y para ello, usan al protocolo OpenFlow, que es un elemento fundamental para la construcción de soluciones SDN.

La arquitectura SDN tiene como características principales:

- Es directamente programable: Se puede tener el control de la red mediante programación, ya que está desacoplada de las funciones de redirección.
- Gestión de forma centralizada: La inteligencia de la red está centralizada en un software basado en controladores SDN que mantienen una visión global de la red, que aparece en aplicaciones y motores de políticas como un único switch lógico.
- Es Ágil: Abstrayendo el control de desvío permite a los administradores ajustar dinámicamente el flujo de tráfico de toda la red para satisfacer las necesidades cambiantes.
- Basados en estándares abiertos y de proveedor neutral: Cuando se implementa a través de estándares abiertos, SDN simplifica el diseño de la red y la operación porque las instrucciones son proporcionados por los controladores SDN en lugar de múltiples dispositivos y protocolos específicos de los fabricantes.
- Programación configurada: SDN permite a los administradores de red configurar, administrar, asegurar y optimizar los recursos de red muy rápidamente a través de los programas de SDN dinámicos, automatizados, que pueden escribir ellos mismos ya que los programas no dependen de software propietario.

1.1.3 FUNCIONAMENTO

Como se ha mencionado, la tecnología SDN separa los planos de control y datos de los equipos y adicionalmente agrega una capa de aplicación en la que se permite desarrollar aplicaciones para el controlador que determinarán el comportamiento de toda la infraestructura de la red.

En la Figura 1.1 se presenta la Arquitectura SDN con las diferentes capas que la conforman.

Cada una de las diferentes capas de la arquitectura cumple una función específica que se detalla a continuación:

La capa de Aplicación, también conocida como capa de negocio, permite que las diferentes aplicaciones especifiquen los recurso y comportamiento que se requieren de la red, dichas aplicaciones, pueden llamar a servicios externos e inclusive organizar varios controladores para poder cumplir con sus requerimientos.

La capa de control, es la encargada de establecer las políticas de reenvío de paquetes, esto llevará a cabo tareas de enrutamiento, conmutación, aislamiento, ingeniería de tráfico, etc. Debe además tomar decisiones en base a la topología completa de la red y deberá configurar cada dispositivo de red para que cumpla las funciones requeridas en cada momento.

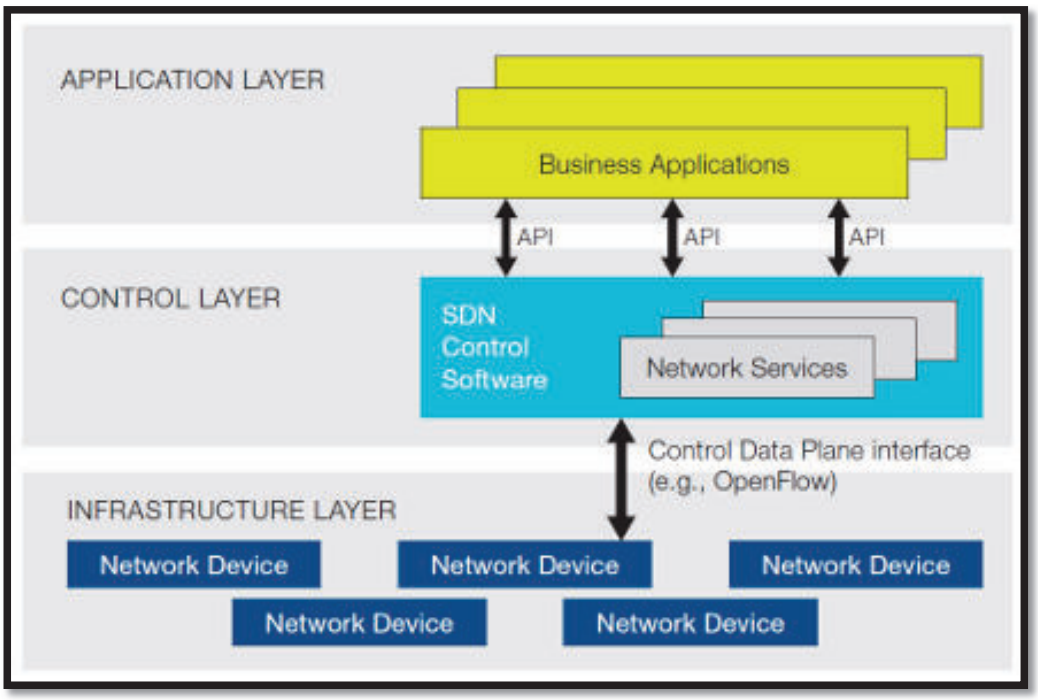

Figura 1.1 Arquitectura SDN [1]

La capa de infraestructura está conformada por todos los dispositivos de red, es la encargada de interactuar directamente con el tráfico de la red, posee únicamente los recursos necesarios para asegurar una correcta virtualización, conectividad, seguridad, disponibilidad y calidad del tráfico de la red.

No dispone de ninguna mecanismo de toma de decisión respeto a los datos, este plano realizará únicamente procesamiento y entrega de paquetes en función de las políticas que el plano de control le indique, es por dicha razón que el control de las funciones de envío y procesamiento ahora se desarrollarán en el controlador.

1.1.4 SDN vs REDES TRADICIONALES

El desarrollo acelerado de nuevas tecnologías que hacen uso de la infraestructura de red de datos como por ejemplo los dispositivos móviles, los servidores de aplicaciones o de datos, la virtualización de equipos, e inclusive servicios en la nube, han llevado a la industria a tener que volver a examinar la estructura de las redes actuales.

El incremento imparable de nuevos servidores de aplicaciones, base de datos, etc., e inclusive la distribución de carga entre estos, hace necesario el envío de información de un lugar a otro para cumplir una determinada tarea antes de llegar al usuario final; así como los servicios en la nube requieren una escalabilidad flexible tanto para procesamiento como para almacenamiento, también las tecnologías de red tradicional necesiten incorporar dicha flexibilidad, y por lo que deben ser mejoradas, dichas tecnologías poseen ciertas limitaciones respecto a las SDN pues no fueron creadas pensando en los requerimientos de los usuarios, empresas y aplicaciones de hoy. A continuación se mencionarán varias de esas limitaciones:

- **Complejidad**: Esto es debido a que en las redes se han creado un sin número de protocolos independientes que resuelven un problema en particular, pero que al ser diferentes unos de otros y debido a la gran cantidad de problemas presentes, cada uno aporta una solución particular pero a la vez complejidad al manejo de las redes actuales. Esta variedad de protocolos muchas veces pueden ser el arma de atacantes para inmiscuirse en la red y realizar un ataque.
- **Políticas inconsistentes:** Actualmente la configuración de una política determinada implica configurar miles de dispositivos y mecanismos. Por ejemplo cuando se agrega un nuevo host, llevaría horas configurar las ACL[4]

[4] ACL (*Access Control List*) es un concepto de seguridad informática que permite determinar que tráfico se debe dejar pasar y que tráfico se debe bloquear.

que permitan darle a ese determinado host las políticas que requiera. La complejidad de las redes actuales, dificulta aplicar políticas de acceso, seguridad, QoS[5], etc. lo que conlleva que las redes queden vulnerables ante ciertas brechas de seguridad.

- **Dependencia de los fabricantes:** Cada fabricante desarrolla su propia solución a un mismo problema lo que hace que muchos fabricantes destinen recursos y tiempo para obtener una solución a un mismo problema. Con las redes SDN las aplicaciones serán independientes del equipo y simplemente bastará con adquirir la aplicación necesaria para cumplir una determinada función, esto disminuye la sobrecarga de los equipos al manejar una gran cantidad de protocolos que mayormente no usan en las funciones para las cuales son utilizados.

1.2 HONEYNETS

Tradicionalmente, la seguridad de los sistemas informáticos se la ha tratado desde un solo punto de vista, el defensivo, se pensó desde un inicio en mecanismos para cifrar información, luego en implementar firewalls, y posteriormente IDS[6], etc. siempre tomándolo desde la perspectiva de contrarrestar cualquier ataque que se detecte, sin embargo podría más bien decirse que desde el momento en que se definió únicamente esa postura, se creó uno de los más importantes puntos débiles de la seguridad de redes.

Una nueva tendencia en la seguridad informática y que se encuentra aún en desarrollo, se conoce como *honeynet*, esta podría poner fin a esa debilidad y más bien sacar provecho de ella y obtener mejores resultados.

1.2.1 DEFINICIÓN

Una *honeynet* es una red con vulnerabilidades previamente configuradas, dispuesta como un blanco de ataque, de esta forma atrae a posibles atacantes, ya

[5] QoS (*Quality of Service*) es el rendimiento promedio de una red de datos desde el punto de vista del usuario, para determinarla de forma cuantitativa, se deben considerar varios aspectos del servicio de red.

[6] IDS (*Intrusion Detection System*) es un conjunto de programas para detectar y recopilar información de accesos no autorizados dentro de una red.

que a diferencia de lo que se cree, la mayor cantidad de ataques se las hace a sistemas con vulnerabilidades fáciles de penetrar.

Este sistema, ha sido diseñado para recolectar información de posibles amenazas, sin embargo su objetivo no está completamente definido pues depende del uso que se les dé, por ejemplo, puede ser para la recolección de información cuando se la usa con propósitos de investigación, o se la puede usar para detectar e inclusive engañar atacantes, cuando el propósito es evitar un ataque a un posible servidor real, u otros objetivos más dependiendo si se las usa con fines militares, políticos, gubernamentales, etc. Una *honeynet* no es un solo sistema sino una red de sistemas que pueden contener una serie de dispositivos de red, que pueden ser de diferentes fabricantes por ejemplo routers CISCO, switch HP, etc., inclusive pueden ser los mismos equipos ejecutando diferentes sistemas operativos como por ejemplo servidores LINUX, servidores WINDOWS, etc.

Los sistemas que conforman la *honeynet* son sistemas comerciales estándar, y corren aplicaciones reales instaladas de forma tradicional y con configuraciones por defecto, intentando, por supuesto, no mostrar que se trata de un sistema señuelo [2].

Debido a su mayor complejidad por el hecho de tratarse de varios equipos, también aumenta el trabajo, el riesgo y esfuerzo para su administración [3].

1.2.2 CARACTERÍSTICAS

Entre las principales características de las *honeynets* se destacan las siguientes [4]:

- Reducción dramática de falsos positivos, por definición cualquier interacción con la *honeynet* es no autorizada, por lo tanto se trata de una ataque.
- Captura y controla todo el tráfico que se dirige hacia los equipos dentro de su infraestructura para su posterior análisis.
- Requiere de la implantación de mecanismos para: captura, control, recolección y análisis de datos.
- Recolección de datos únicamente cuando alguien esta interactuando con ella, esto conlleva a un posterior análisis mucho más fácil, pues los datos únicamente se recolectaron durante el tiempo de actividad y no durante todo

el tiempo. Esta característica la vuelve mucho más efectiva que el uso de un IDS para la detección de amenazas y captura de datos.

- Puede implementarse usando diferentes arquitecturas, ya sean propias o previamente establecidas como las de Generación I y Generación II que se estudiarán más adelante.
- Poseen problemas asociados con el aspecto legal, pues pueden comprometer más sistemas de los que se intenta.

1.2.3 VENTAJAS

Los principales beneficios de trabajar con *honeynets* en cuanto a implementación de mecanismos de seguridad, que podrían permitir posteriormente desarrollar procedimientos para contrarrestar ataques de terceros a redes corporativas e inclusive redes pequeñas como las que se tiene en casa, se podrían resumir en los siguientes:

- Disuadir ataques
- Detección temprana de intrusos
- Descubrimiento de nuevos ataques
- Notificación temprana
- Gestión de logs

1.2.3.1 Disuadir Ataques

Cuando un atacante se da cuenta de que el sistema en el que se ha infiltrado es una red en la que lo están monitoreando, simplemente desiste de continuar en dicha red y opta por abandonarla, cabe resaltar que ese sin embargo no es el objetivo de una *honeynet*.

Siempre que las acciones tomadas por el atacante no afectan información importante de la red real, sino únicamente a la información de la *honeynet*, se puede hablar de disuadir el ataque, ya que al no darse cuenta que está en la *honeyne*t sus acciones no representan peligro, pero si son importantes para su posterior análisis.

1.2.3.2 Detección temprana de intrusos

Desde su concepción, se debe saber que una *honeynet* es una red señuelo (una trampa), por lo tanto cualquier tráfico que se detecte viajando desde o hacia la *honeynet* se presume es malicioso inmediatamente se lo detecte.

1.2.3.3 Descubrimiento de nuevos ataques

La concepción principal de una *honeynet* es recoger información, a pesar de lo que ya se ha mencionado sobre posibles usos adicionales, es por tanto una tecnología que permite conocer al enemigo para posteriormente buscar formas de contrarrestar sus ataques, o incluso registrar que información es la más atrayente y vulnerable.

1.2.3.4 Notificación temprana

Al tratarse de un sistema señuelo, inmediatamente al detectar tráfico entrante o saliente a la *honeynet*, se puede alertar al administrador de red y seguidamente tomar las acciones pertinentes para mantener el ataque en la *honeynet* y reforzar las defensas de la red, la forma de notificación, puede ser implementada de diferentes formas, por ejemplo mediante llamadas telefónicas, envío de mensajes de texto, correo electrónico, etc.

1.2.3.5 Gestión de logs[7]

Si se es lo bastante hábil para permitir al atacante realizar las acciones que intento, sin que se percate de que estuvo siendo monitoreado y bajo control absoluto, se podrá gestionar los logs que se registraron durante la intromisión del mismo y determinar su modo de operación [5].

1.2.4 DESVENTAJAS

A pesar de ser numerosas las ventajas que se pueden obtener al utilizar esta tecnología, también se tiene ciertas desventajas que pueden limitar el uso de estas o también en algunos casos limitar su potencial, entre los principales factores negativos con los que se puede encontrar al ponerlas en funcionamiento están las siguientes:

- Identificación

[7] Logs son registros, bitácoras de cada una de las actividades realizadas en los equipos, en los que se puede recolectar direcciones IP, horas, feches, etc.

- Riesgo
- Recursos
- Administración

1.2.4.1 Identificación (*fingerprinting*)

Se refiere a la identificación de la identidad de un elemento de la *honeynet* debido a ciertos patrones y características que delatan su propósito. Si un intruso detecta la presencia de un *honeypot*[8] en alguna organización, este podría utilizar la identidad de otro equipo en la organización para atacarlo y de esta manera cuando el administrador de la red se dé cuenta, iniciará una investigación interna, mientras el atacante se concentra en equipos reales. Además si un intruso identifica un *honeypot* de investigación este podría generar datos que desvíen la investigación y resulten en conclusiones erróneas sobre el comportamiento de la red.

1.2.4.2 Riesgo

Cuanta más cantidad de datos se quiera recolectar mayor es el riesgo, se debe permitir al atacante gozar de ciertos privilegios para realizar algunas tareas sin embargo entre más privilegios se le otorgan, para que no se percate de que se encuentra en una *honeynet* y así permitir analizar su comportamiento malicioso, mayor es el riesgo de que se salga de control y al final el atacante logre terminar con éxito sus propósitos maliciosos. Incluso se debe tener cuidado de no dejar demasiadas vulnerabilidades y a su vez percatarse que aquellas que se hayan dejado, no sean demasiado obvias para que el atacante no sospeche.

1.2.4.3 Recursos

Para que la red que se implemente tenga una mayor utilidad y sea un mejor señuelo, se deben incorporar en ella una serie de equipos en los que se corran servidores reales, incluyendo hosts de usuarios, los equipos usados deben ser en lo posible de diferentes fabricantes y corriendo distintos servicios en diferentes plataformas.

---

[8] *Honeypot* es un sistema vulnerable real o virtual que está listo para ser atacado, generalmente cada equipo que conforma una *honeynet* es un *honeypot*.

La implementación de la *honeynet*, la cual preferiblemente se la realiza de forma virtual, ocupa recursos tanto de hardware como de software. Lo que se traduce en gastos económicos y administrativos. Siendo factores importantes que se toman en cuenta en la decisión de implementar este método.

1.2.4.4 Administración

El implementar la *honeynet* con todos sus equipos y servicios corriendo, es únicamente el primer paso, es decir desde este punto empieza el trabajo, no vale la pena simplemente construir una red y dejarla abandonada esperando que lleguen atacantes y se recolecten datos. Se debe monitorear de forma periódica para ver si existen posibles amenazas, se deben tomar los datos obtenidos y convertirlos en información útil mediante técnicas efectivas de análisis y aún más se deben instalar actualizaciones y parches en los dispositivos.

1.2.5 ARQUITECTURA

La arquitectura con la que se despliegue la *honeynet* dependerá del propósito y de los conocimientos que se tenga, es importante que se utilice únicamente la arquitectura determinada en la Generación I cuando no se disponen de conocimientos lo suficientemente amplios, pues su utilización resulta mucho más segura que la de Generación II, aunque menos eficiente. La implementación de la arquitectura de Generación II, implica ciertas definiciones y conceptos de mucha más complejidad que hacen indiscutiblemente más difícil su control. Los resultados obtenidos usando la arquitectura determinada por la Generación II, son mucho más útiles pero también lo es su grado de riesgo.

A continuación se detallan dos arquitecturas que han sido desarrolladas por "*The Honeynet Project*"[9], y que actualmente continúan siendo utilizadas.

1.2.5.1 Generación I

La primera generación está marcada significativamente por una pieza clave que se trata de un *honeywall* de capa tres, que es el elemento principal para controlar el tráfico permitiendo todas las conexiones entrantes pero bloqueando ciertas conexiones salientes.

---

[9] *The Honeynet Project* es una organización internacional dedicada a la investigación en el campo de la seguridad informática.

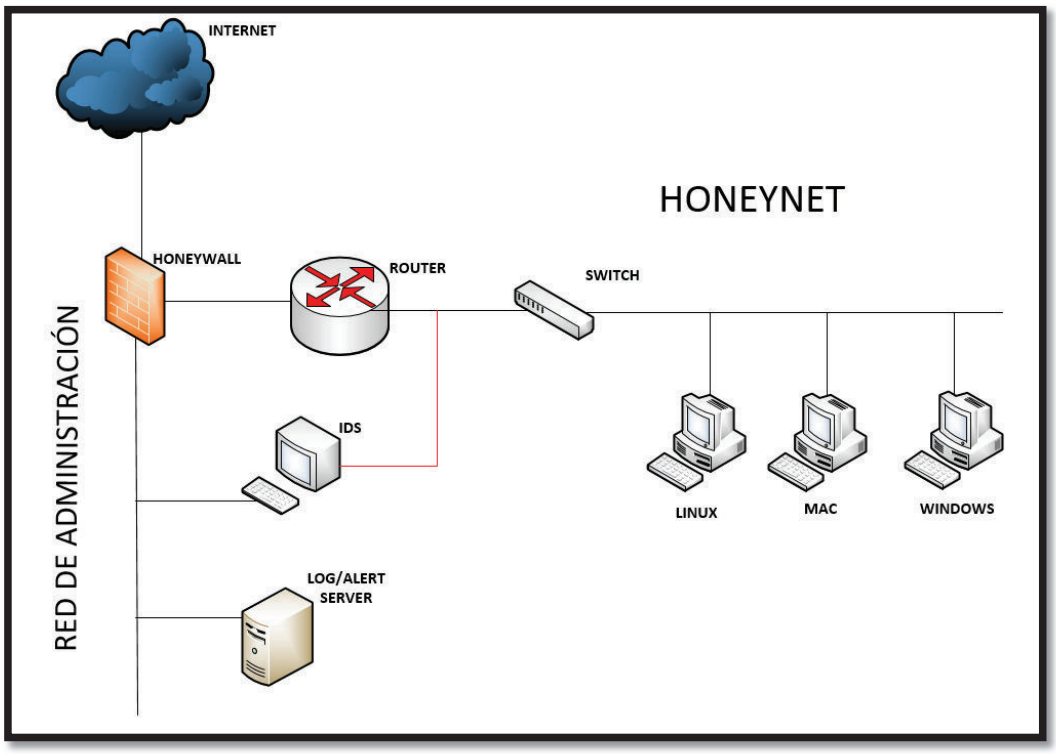

Figura 1.2 Estructura de una *honeynet* de Generación I

El *honeywall* mantiene un control del número de conexiones salientes, cuando llegan a un límite previamente establecido se bloquean. El router se coloca entre la *honeynet* y el *honeywall* para ocultarlo y para dar un control adicional de los datos que circulan, como se muestra en la Figura 1.2, el *honeywall* dispone de 3 interfaces de red (interna, externa y administración), la interfaz externa es usada para conectarse a Internet, la interfaz interna para conectarse a la *honeynet* y la última para conectarse con el servidor de logs.

Para la captura de datos se dispone de un IDS que tiene dos funciones principales, la primera es la captura de todos los datos que circulan por la red y la segunda es alertar al administrador cuando un evento es generado por un atacante.

El IDS posee dos interfaces, una de ellas tiene una dirección IP[10] y es utilizada para el manejo y recolección de datos, la otra no posee dirección IP, y es por donde se realiza la captura de datos, por lo que al ser configurada de esta forma es más difícil

[10] Dirección IP es una etiqueta numérica que permite identificar de forma lógica y jerárquica una interfaz física de un equipo en la red.

de detectar y atacar. El *honeywall* también es usado para este propósito, debido a que por el pasan todas las conexiones entrantes o salientes, y este mantiene información sobre estas.

1.2.5.2 Generación II

Fue introducida en el año 2002 y busca solventar muchos de los problemas que aparecieron en la Generación I, esta arquitectura es más fácil de implementar, difícil de detectar y de mantenimiento más seguro.

En esta generación, se introduce un dispositivo al que se denomina "*Honeywall Gateway*" que es la puerta de enlace a la *honeynet*, dicho dispositivo combina en un solo equipo los elementos de *honeywall* e IDS mostrados en la Generación I separado.

Las tareas de control y captura de datos ahora están centralizadas en un solo dispositivo que trabaja a nivel de capa 2 de acuerdo al modelo ISO/OSI llamado *Honeywall Gateway* como se muestra en la Figura 1.3. Este método, muy común en este tipo de mecanismos, permite prescindir de dirección IP, reduciendo las posibilidades de detección por parte de los atacantes, además al tratarse de un solo equipos esta arquitectura es más fácil de desarrollar y mantener.

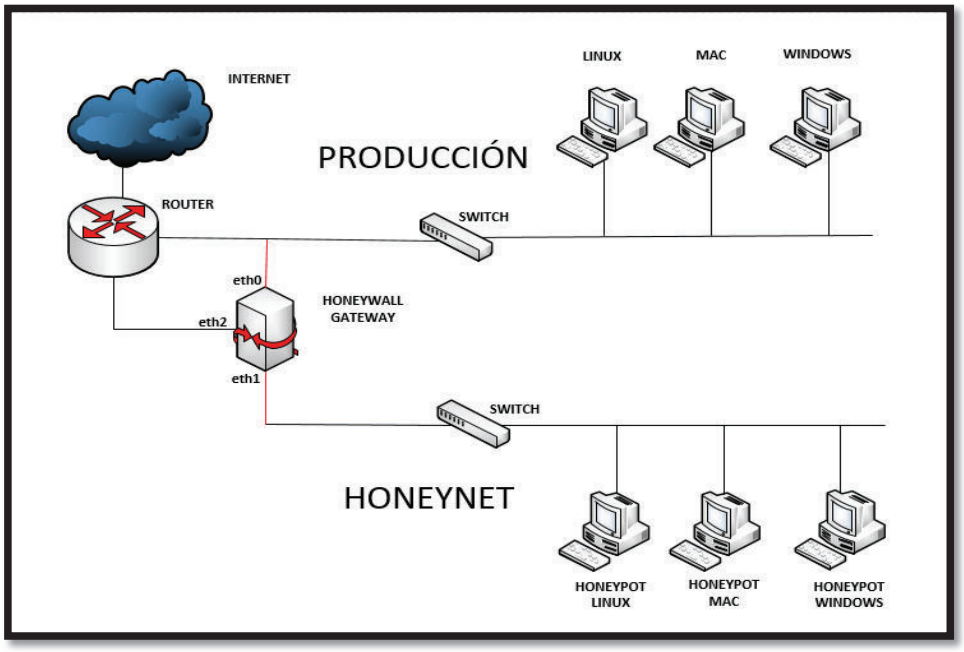

Figura 1.3 Estructura de una *honeynet* de Generación II

El *Honeywall Gateway* es el componente más crítico de toda la arquitectura, este posee 3 interfaces de red, donde se conectan: la red de producción, la *honeynet* y la interfaz de administración; las interfaces eth0 y eth1 trabajan únicamente a nivel de capa 2, pues este dispositivo trabaja como puente, esto ayuda a que se integre la *honeynet* dentro de la red de producción, sin embargo la interfaz eth2 trabaja a nivel de capa 3 y es utilizada para la administración del equipo.

La Generación II busca aumentar la interacción con el atacante, para ello se quiere darle una mayor flexibilidad permitiéndole un número limitado de conexiones.

Para el control de tráfico no solo se limita a verificar el número de conexiones salientes y se procede a bloquearlas, si no se hace un seguimiento para determinar qué tipo de tráfico es y a quién va dirigido. Sin embargo si llega a efectuar su ataque se lo permite siempre hasta cierto límite ante el cual, el ataque no tenga éxito y de esta manera hacerle creer que su ataque efectivamente se realizó pues se recibieron respuestas, sin embargo no sabrá porque el ataque no surtió efecto.

La captura de datos se lleva a cabo desde el núcleo del Sistema Operativo. Esto permite la captura de tráfico, que con *sniffers*[11] o cortafuegos como en la Generación I no son efectivos [2].

1.2.6 REQUERIMIENTOS DE IMPLEMENTACIÓN

Es necesario que una *honeynet* cumpla ciertos requerimientos que ayuden a cumplir la funcionalidad para la que se la ha implementado. Es por eso que debe implementarse ciertos mecanismos que permitan cumplir funciones determinadas, varias de las cuales ya se han sido mencionadas anteriormente cuando se revisó las arquitecturas de las *honeynets*, entre las que se tiene:

- Control de datos
- Captura de datos
- Recolección de datos
- Análisis de datos

[11] *Sniffer*: es una herramienta que permite capturar un paquete enviado por la red.

La recolección de datos, es un requisito que no todas las *honeynets* deben cumplir sino únicamente aquellas que posean varias *honeynets* de forma distribuida y por lo tanto necesiten recolectar la información de forma centralizada para su posterior análisis.

1.2.6.1 Control de Datos

El control de datos hace referencia a la forma en que se gestiona el tráfico de terceros a la LAN[12], así como el tráfico que se envíe desde esta hacia Internet por parte de los mismos, es decir cualquier intento de conexión desde o hacia afuera debe ser permitido o rechazado de forma planificada. Esto deberá implementarse y controlarse de forma tal que no se permita generar ataques desde la LAN una vez que el sistema haya sido comprometido, tampoco se debería limitar totalmente los intentos de conexión hacia el exterior pues sería evidente para el atacante que se trata de un sistema señuelo al que ha comprometido y procederá a dejarlo, no antes intentando eliminar los registros que pudo haber generado.

1.2.6.2 Captura de Datos

Una vez que se ha sido determinado el tráfico entrante y saliente que se admitirá en la LAN, se debe implementar mecanismos que permitan capturar dicho tráfico.

En sistemas UNIX[13] la captura de datos y monitoreo se puede realizar utilizando un servidor rsyslog[14], el cual permite guardar los datos de forma centralizada y remota, debido a que no conviene de ninguna manera mantener la información almacenada en el mismo sistema comprometido, ya que el atacante podría percatarse y borrarla. Aun si el atacante se percata de que existe un servidor de logs, es mucho más difícil vulnerarlo y por lo tanto necesitará mucha hábilidad para lograrlo, sin embargo si logra también comprometerlo y borra la información contenida en este, el cortafuego de la *honeynet* , el *honeywall* también deberá haber registrado el tráfico que circundo y ciertas acciones o comandos que se hayan ejecutado, por lo que se tendrá un respaldo de los datos capturados a pesar de que no sea tan minucioso como el que se obtuvo en los servidores de logs. En

---

[12] LAN (*Local Area Network*) es una red que conecta los dispositivos en un área relativamente pequeña y predeterminada (como una habitación, un edificio, o un conjunto de edificios).
[13] UNIX es un sistema operativo multiusuario, multitarea, con un entorno para trabajo en red, ofrece programas y servicios.
[14] RSYSLOG es una utilidad de software usado en UNIX para enviar mensajes log en una red IP.

sistemas Windows también existen aplicaciones que pueden permitir recolectar información y enviarlas a un servidor remoto.

1.2.6.3 Recolección de Información

Es preferible tener un servidor centralizado en caso de que dentro de la institución se tenga implementadas varias *honeynets*, por lo tanto se debe tener un mecanismo que permita reunir la información de los diferentes servidores de logs de cada una de las *honeynets* en un servidor central. El análisis del conjunto de datos obtenidos de las diferentes *honeynets* que se realizará posteriormente puede brindar más información que la que se obtiene analizando los datos capturados de forma individual en cada *honeynet*.

1.2.6.4 Análisis de Datos

Debido a que el objetivo principal de una *honeynet*, es meramente para fines investigativos, se deben analizar las acciones y el modo de operación de atacantes, si se llegara a desviar de este propósito, o si no se obtiene la suficiente información, la *honeynet* carece de importancia o de utilidad. Por lo tanto se debe tener un buen mecanismo que permita analizar los datos que se recolecten para poder obtener información útil para el propósito con el cual se la creo. Un mecanismo que podría ser útil es el análisis descriptivo, que ayuda a observar el comportamiento de la muestra en estudio mediante tablas, gráficos, etc.

1.2.7 ESTADO DEL ARTE DE LAS HERRAMIENTAS DE DESPLIEGUE

Entre las principales herramientas que permiten desplegar el uso de *honeynets*, se puede mencionar algunas que ayudan a implementar de forma mucho más sencilla los requerimientos que una *honeynet* debe tener, estas herramientas se encuentran disponibles en la página oficial de "*The Honeynet Project*".

1.2.7.1 Herramientas de Despliegue

Las herramientas de despliegue han sido desarrolladas por "*The Honeynet Project"* para poder implementar de una manera mucho más sencilla una *honeynet*. Todas

estas herramientas son de código abierto bajo la licencia BSD[15] a menos que se especifique otra cosa.

1.2.7.2 Tipos de Herramientas:

Las diferentes herramientas desarrolladas que se mencionarán están clasificadas de acuerdo al requerimiento que cumplen en una *honeynet*, es decir se mencionarán en primer lugar aquellas de utilidad para el control de datos, luego aquellas para la captura de datos, posteriormente a las herramientas útiles para la recolección de datos y finalmente las que ayudan a realizar el análisis de los mismo [6].

1.2.7.2.1 Control de Datos

Entre las herramientas que permiten realizar control de los datos en la *honeynet* se encuentra las siguientes:

- **Rc.firewall:** Se trata de un script para *IpTables*[16] que permite llevar en cuenta el número de conexiones que se generen y de ser el caso bloquear algunas de las conexiones salientes en sistemas Linux. La nueva versión permite trabajar con las arquitecturas tanto de Generación I como de Generación II.
- **Snort_inline**: Herramienta que permite bloquear o modificar ciertos ataques conocidos cuya forma de actuar es predecible. Es una versión modificada de *snort*, que acepta paquetes de *IpTables*, y los procesa (permite, bloquea, rechaza, elimina, etc.) de acuerdo al conjunto de reglas de *snort*, puede pensarse como un IPS[17] que usa un IDS existente para el manejo de los paquetes.
- **Ebtables**: Es un programa que se usa como herramienta de filtrado en un firewall bridge que esté basado en Linux. Este habilita un filtrado transparente para el tráfico que pasa a través del bridge Linux, principalmente está destinado a trabajar en la capa de enlace de datos del

[15] Licencia BSD es una licencia de software libre permisiva con mucho menos restricciones que otras, como por ejemplo la GPL. Adicionalmente permite el uso de código fuente en software no libre.
[16] IpTables es una herramienta de cortafuegos que permite no solamente filtrar paquetes, sino también realizar traducción de direcciones de red (NAT) para IPv4 o mantener registros de log.
[17] IPS (*Intrusion Prevention System*) es un software que ejerce el control de acceso en una red informática para proteger a los sistemas computacionales de ataques y abusos.

modelo de referencia OSI, a pesar de que puede aplicar ciertos filtros básicos en capas superiores [7].

1.2.7.2.2 Captura de Datos

Para implementar este requerimiento de una *honeynet*, se pueden utilizar herramientas como las que se mencionan a continuación:

- **Sebek**: Esta herramienta consta de un cliente que está en la *honeynet* sin que se percate el atacante, analizará los datos ya que monitoriza constantemente y los datos de interés los enviará al servidor de forma oculta, estos datos se guardarán para un posterior análisis del administrador.
- **Bash Patch**: Es un parche que modifica el shell `/bin/bash`, esto permite enviar usando el protocolo UDP[18] todas las pulsaciones que se hagan en el teclado, para que sean capturados mediante un *sniffer* y luego almacenados en un servidor de logs.

1.2.7.2.3 Recolección de Datos

La relación de datos se puede implementar utilizando herramientas tales como:

- **Upload Script**: Se trata de un *script* que permite subir a una base de datos toda la información que se recolecte de forma diaria. Este *script* sube dos conjuntos de datos, los registros del *firewall* de *IpTables* y los ficheros de registro binarios de *snort*.
- **Obfugator:** Se trata de una herramienta utilizada para sanear los registros de sistema, que incluye no solo las cabeceras de los paquetes, sino también datos incluidos en los mismos.

1.2.7.2.4 Análisis de Datos

Entre las herramientas que permitirán realizar un análisis los datos que han sido capturados en la *honeynet* se tiene:

- **Honeysnap:** Es una herramienta usada para extraer y analizar datos de archivos pcap[19]. Esta herramienta usa una línea de comandos para analizar

[18] UDP (*User Datagram Protocol*) es un protocolo de transporte de datos en redes IP.
[19] PCAP es una interfaz de una aplicación de programación para captura de paquetes

uno o varios archivos a la vez y produce un reporte del análisis que identifica eventos relevantes del procesamiento de datos [8].

- **Privmsg**: Se trata de un script desarrollado en Perl[20] que permite extraer conversaciones IRC[21] de ficheros de registro binarios de tcpdump[22], muy bueno para eliminar "ruido" o "basura".
- **WinInterrogate:** Se trata de un conjunto de soluciones de código abierto que permiten realizar el análisis de sistemas de ficheros y procesos win32.

1.2.8 ANÁLISIS COMPARATIVO DE LAS HERRAMIENTAS DE DESPLIEGUE

Actualmente existen muchas herramientas que permiten implementar de forma mucho más sencilla una *honeynet*, muchas de estas herramientas (las más conocidas) han sido ya mencionadas en el punto anterior, a continuación se presenta un análisis comparativo entre dichas herramientas de acuerdo a la función que cumple cada una de ellas:

1.2.8.1 Control de Datos

Las características de las herramientas que permiten realizar control de los datos en la *honeynet* se encuentran presentadas en la Tabla 1.1 .

CARACTERISTICA \ HERRAMIENTA	Rc.firewal	Snort_inline	Ebtables
Herramienta basada en Linux	SI	SI	SI
Trabajo solo con una dirección IP estática para salir a Internet	SI	NO	NO
Controla el número de conexiones	SI	SI	SI
Permite bloquear/aceptar conexiones	SI	SI	SI
Realiza modificación de ataques conocidos	NO	SI	NO
Permite rechazar/eliminar conexiones	SI	SI	SI
Realiza filtrado de paquetes	SI	SI	SI
Enrutamiento tráfico usando direcciones físicas	NO	NO	SI
Capa de trabaja según el modelo OSI	3	3	2, 3
Realiza manejo de paquetes	NO	NO	SI

[20] Perl es un lenguaje de programación basado en C, que trabaja a un grado inferior a otros lenguajes.
[21] IRC (*Internet Relay Chat*) protocolo en tiempo real basado en texto.
[22] TCPDUMP es una herramienta en línea de comandos cuya utilidad principal es analizar el tráfico que circula por la red.

Posee una licencia GPL[23]	SI	SI	SI

Tabla 1.1 Diferentes herramientas para control de datos en una *honeynet*

1.2.8.2 Captura de Datos

La captura de datos en una *honeynet* puede realizarse haciendo uso de las herramientas que han sido mencionadas anteriormente, se presenta en la Tabla 1.2 una comparación entre las características que estas poseen.

CARACTERISTICA \ HERRAMIENTA	Sebek	Bash Patch	Termlog
Herramienta basada en Linux	SI	SI	SI
Captura pulsaciones de teclado	SI	SI	NO
Trabaja con un servidor centralizado	SI	NO	SI
Trabaja como *sniffer*	NO	SI	NO
Cifra la información capturada antes de enviarla	NO	NO	SI
Realiza procesamiento básico de información	SI	NO	SI
Puede trabajar en Windows	SI	NO	NO
Es un módulo del *kernel*	SI	NO	SI
Basado en la arquitectura Cliente-Servidor	SI	NO	NO

Tabla 1.2 Comparación entre las diferentes herramientas para captura de datos en una *honeynet*

1.2.8.3 Recolección de Datos

En la Tabla 1.3, se presenta una comparación entre las características relevantes de las herramientas más usadas para realizar la tarea de recolección de datos.

CARACTERISTICA \ HERRAMIENTA	Upload Script	Obfugator
Herramienta basada en Linux	SI	SI
Envía información a un servidor	SI	NO
Trabaja en conjunto con un firewall	SI	NO
Encripta la información capturada antes de enviarla	NO	NO
Posee una Licencia GPL	SI	SI

Tabla 1.3 Comparación entre las diferentes herramientas para recolección de datos en una *honeynet*

[23] GPL (*General Public License*) es una licencia pública usada en software.

1.2.8.4 Análisis de Datos

Las características más relevantes de las herramientas más utilizadas para realizar el análisis de datos se presentan en la Tabla 1.4.

CARACTERISTICA \ HERRAMIENTA	Honeysnap	Privmsg	Sleuthkit	WinInter
Herramienta basada en Linux	SI	SI	SI	NO
Realiza análisis estadísticos	SI	NO	NO	NO
Permite eliminar información dañada	NO	SI	NO	NO
Permite mostrar información gráfica	NO	NO	NO	SI
Permite además recolectar información	NO	NO	NO	SI
Licencia GPL	SI	SI	SI	NO

Tabla 1.4 Comparación entre las diferentes herramientas para análisis de datos en una *honeynet*

1.3 HERRAMIENTAS DE ATAQUE

Actualmente existe una gran variedad de herramientas informáticas que permiten generar diferentes tipos de ataques, inclusive existen distribuciones de Linux que han sido desarrolladas explícitamente para incorporar un gran número de dichas herramientas como por ejemplo BackTrack o Kali.

A continuación se detallan dos de estas herramientas que se consideran de gran utilidad para el desarrollo de los diferentes ataques que se emplearán durante el desarrollo del proyecto.

1.3.1 ETTERCAP

Es una de las herramientas más poderosas para interceptar comunicaciones de área local e inalámbricas, nació como un *sniffer* que permite capturar paquetes que circulan por redes conmutadas, sin embargo durante su desarrollo se han incorporado varias características adicionales que lo han convertido en una poderosa y flexible herramienta para realizar ataques informáticos, adicionalmente se han incorporado herramientas que posibilitan analizar los diferentes protocolos, e incluye muchas otras características para analizar tanto la red como los hosts. Entre sus principales funciones se pueden mencionar las siguientes:

- Inyección de paquetes en una comunicación establecida previamente y que a un está activa

- Compatibilidad y soporte a SSH[24], SSL[25] y HTTPS[26]
- Soportes para las diferentes funcionalidades
- Ataques contra túneles PPTP[27]
- Detección de sistema operativo remoto
- Filtrado y sustitución de paquetes [9]
- Recolección de contraseñas para protocolos de texto plano
- Finalización de conexiones [10]

1.3.2 HPING

Es un ensamblador de paquetes IP[28], orientado a trabajar con el protocolo TCP[29]. Se trata de una línea de comandos similar a la línea de comandos de UNIX, sin embargo es capaz de enviar paquetes ICMP[30], TCP, UDP, otros. Todos los campos de las cabeceras de las PDU[31] creadas pueden ser modificados y controlados usando la línea de comandos. Se utiliza como herramienta de seguridad para las redes y para análisis de los mismos protocolos.

También incorpora una habilidad importante como lo es el permitir enviar archivos a través de Internet para lo cual se necesita un equipo que envíe el archivo y otra que este escuchando para recibirlo. De acuerdo al uso anteriormente mencionado es posible entonces utilizarlo como si fuese un troyano.

---

[24] SSH (*Security Shell*) es un protocolo y programa que permite acceder de forma remota a través de una red.
[25] SSL (*Security Socket Layer*) protocolo que permite una comunicación segura.
[26] HTTPS protocolo usado en la transacción WWW (*World Wide Web*) de forma segura, basado en HTTP.
[27] PPTP protocolo desarrollado por Microsoft para redes privadas virtuales (VPN).
[28] IP (*Internet Protocol*) es un protocolo de comunicación de datos digitales clasificado funcionalmente en la Capa de Red según el modelo internacional OSI.
[29] TCP (*Transmission Control Protocol*) protocolo que permite establecer conexiones para el flujo de datos.
[30] ICMP (*Internet Control Message Protocol*) es un sub protocolo de notificación y control de errores del protocolo IP.
[31] PDU (*Protocol Data Unit*) se utilizan para el intercambio de datos entre unidades disparejas, dentro de una capa del modelo OSI.

Esta herramienta es perfecta para realizar ataques tales como DoS[32], SPOOFING[33] o FLOODING[34] [11].

Entre las tareas que se pueden desarrollar con HPING se tiene:

- Pruebas de firewalls
- Escaneo avanzado de puertos
- Descubrimientos de sistema operativo remoto
- Testeo de redes
- Uso similar a Traceroute [12]

1.4 TECNICAS DE ATAQUE

Una técnica de ataque informático es un método usando por un individuo, para tomar el control, desestabilizar o dañar otro sistema informático o red.

Actualmente existe un gran número de ataque informáticos, sin embargo en este documento se presenta una breve descripción de los ataques que se efectuarán en el proyecto, los cuales son:

- Ataque de denegación de servicio
  - THC SSL DoS
  - TCP SYN Flood
  - Smurf
- Ataques de envenenamiento
  - IP Spoofing
  - ARP Spoofing
  - DNS Spoofing

[32] DoS (*Denial of Service*) es un ataque a una red de computadoras que causa que un servicio o recurso sea inaccesible.
[33] Hace referencia al uso de técnicas de suplantación de identidad generalmente con usos maliciosos o de investigación.
[34] *Flooding* Consiste en enviar mucha información en poco tiempo a alguien para intentar que haga uso de todos los recursos de los que dispone.

1.4.1 ATAQUE DE DENEGACIÓN DE SERVICIO

También conocido como ataque de DoS (*Denial of Service*), es un ataque a una red de computadores que hace que un recurso sea inaccesible a verdaderos usuarios, generalmente genera una pérdida a la conectividad de la red por el consumo excesivo ya sea del ancho de banda de la red o de los recursos de procesamiento del equipo atacado.

Existen algunas variedades entre las cuales se pueden mencionar los siguientes:

1.4.1.1 THC SSL DoS

THC_SSL_DoS es una herramienta para verificar el desempeño del protocolo SSL.

El protocolo SSL requiere mucho más poder de procesamiento en el servidor que en el cliente cuando se realiza una conexión segura, esta cantidad de procesamiento desproporcionada se la toma como un punto débil y se intenta sobrecargar al servidor y dejarlo fuera de servicio.

Cuando se intenta realizar una inundación de paquetes en un servidor mediante DDoS[35], con conexiones basadas en DSL[36], el servidor no se ve afectado pues estos generalmente gozan de un gran ancho de banda, suficiente para mantener conexiones con un gran número de usuarios, es por tanto inútil un ataque bajo estas circunstancias, sin embargo esta herramienta aprovecha el excesivo poder de procesamiento que demanda el *handshake*[37] que se realiza al inicio de una conexión segura en el lado del servidor, generalmente los servidores no están preparados para manejar una gran cantidad de *handshakes*.

El peor escenario de ataque entonces, es un ataque denominado SSL-Exhaustivo, en el que miles de clientes realizan intentos de conexión simultánea de tipo segura al mismo tiempo (SSL_DDoS).

Con el uso de esta herramienta se muestra las vulnerabilidades del protocolo SSL cuando este trabaja con renegociación de conexión principalmente, sin embargo es

---

[35] DDoS (*Distribute Denial of Service*) es un ataque de DoS realizado desde múltiples equipos.
[36] DSL (*Digital Subscriber Line*) es una familia de tecnologías que proporcionan el acceso a Internet mediante la transmisión de datos digitales a través de los cables de una red telefónica.
[37] Handshake es el intercambio de mensajes que se hace en los diferentes protocolos para establecer la comunicación.

posible también trabajar sin esta, simplemente estableciendo una nueva conexión TCP haciendo algunas modificaciones al *script*.

Debe considerarse que en promedio un servidor puede realizar 300 *handshakes* por segundo, lo que podría requerir del 10 al 25% del uso del procesador de un computador para realizar el ataque [13].

Se deben utilizar múltiples host para realizar el ataque en caso de que se esté usando un acelerador de SSL[38] del lado del servidor.

Y finalmente se debe tener muy en cuenta que no siempre el puerto 443 usado para HTTPS es siempre la mejor opción, muchas veces es conveniente usar puertos de protocolos como POP3S[39], SMTPS[40], etc., en cuyos puertos es menos probable el uso de aceleradores de SSL.

1.4.1.2 TCP SYN FLOOD

Este tipo de ataque es posible prácticamente gracias a la forma de funcionamiento del protocolo TCP. Cuando un host requiere iniciar una conexión de este tipo, envía un paquete denominado SYN, el receptor al recibir este paquete determina que alguien le está pidiendo un inicio de conexión y por lo tanto contesta con un paquete denominado SYN/ACK en el que contesta la petición y pide que se le confirme que fue recibida su respuesta, el host que inicio la conexión entonces al recibir dicho paquete responde finalmente con un paquete denominado ACK y esta lista la nueva conexión.

Considerando este intento de conexión generalmente denominado en tres vías, se genera el ataque SYN_FLOOD, que consiste en enviar un gran número de intentos de conexión a una víctima y no contestar el ACK final para que esta, se quede esperando y de esta forma consumiendo recurso que podrían haber sido usados para realizar conexiones con usuarios verdaderos, como se muestra en la Figura 1.4 [14].

---

[38] Acelerador de SSL es el dispositivo que se utiliza para la descarga de los algoritmos de cifrado en las transacciones SSL

[39] POP3S (*Post Office Protocol*) es un protocolo seguro basado en POP3 que permite obtener los mensajes de correo.

[40] SMTPS (*Simple Mail Transfer Protocol*) es un protocolo seguro para SMTP que permite realizar envío de correo electrónico

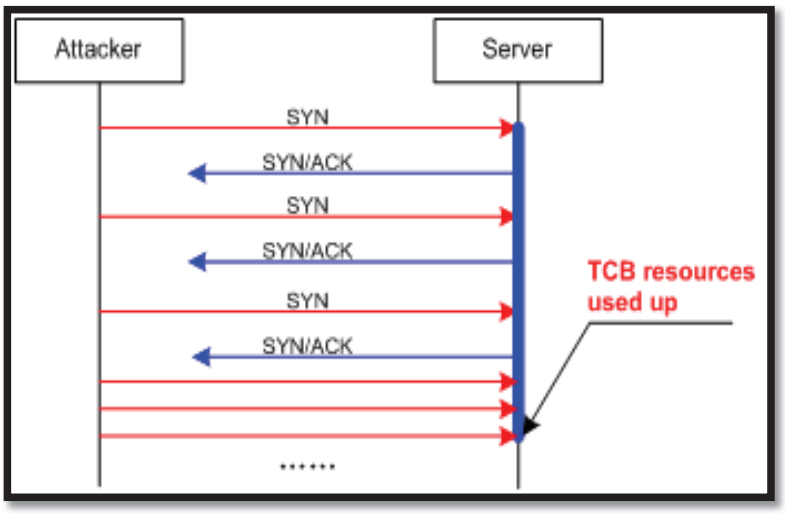

Figura 1.4 Diagrama del ataque TCP SYN FLOOD [15]

1.4.1.3 SMURF

Es otro tipo de ataque del tipo DoS que permite causar que la parte atacada de la red se vuelva inoperable, el atacante crea paquetes ICMP de tipo *echo_request*, con una dirección IP de *broadcast* de destino y con la dirección IP de la víctima como dirección de origen, ocasionando que múltiples equipos (todos a los que se les envió el paquete *broadcast*) contesten a la solicitud y dejen inoperante la víctima [16].

Las víctimas de este ataque pueden ser afectadas directamente o pueden ser parte de una red que se quebranta para amplificar el ataque, como se muestra en la Figura 1.5. Este tipo de ataque generalmente se utiliza para desactivar totalmente la comunicación de un host, sin embargo también puede ser utilizado para reducir la tasa de transferencia de datos o el ancho de banda de modo que la conexión se haga lenta.

Este tipo de ataque se basa principalmente en hacer uso del protocolo ICMP, el cual es empleado legítimamente por el comando denominado PING que se encarga de verificar la conectividad entre dos *hosts*, este tipo de ataque se realiza principalmente poniendo como dirección origen del paquete ICMP la del *router* o *gateway* de la red para que quede inundada completamente y se pierda conectividad con el exterior [17].

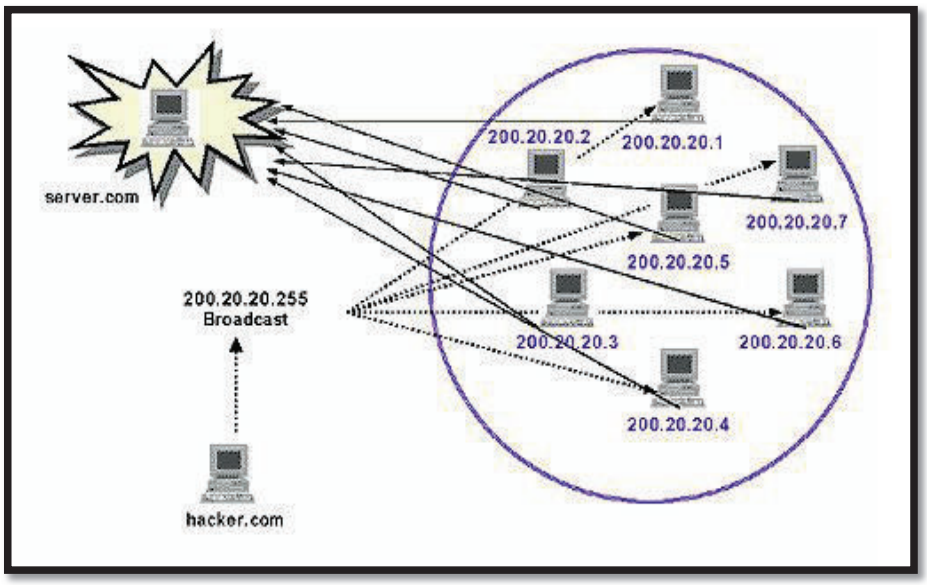

Figura 1.5 Ejemplo de un ataque Smurf [18]

1.4.2 ATAQUES DE ENVENENAMIENTO

Este tipo de ataque se basa principalmente en la creación de tramas TCP/IP en las que se usa una dirección IP falsa, La idea principal de este ataque consiste en suplantar la identidad de otro usuario o host de la red para lograr acceder a recursos de otro sistema que ha establecido algún mecanismo de confianza basado en la dirección IP del *host* suplantado.

Entonces podría decirse que para este tipo de ataque se necesita de la presencia de tres usuarios: el atacante, el atacado y el suplantado. Se debe intentar acceder a información que posee el atacado y a la vez impedir que el equipo suplantado interfiera en dicho proceso, para ello sería de utilidad utilizar uno de los ataques de DoS que se revisaron anteriormente.

A continuación se mencionarán algunos de los tipos de ataques de envenenamiento (*spoofing*) existentes:

1.4.2.1 IP SPOOFING

Es un método de ataque que consiste principalmente en la falsificación de direcciones IP, debido a que se trata de una técnica de secuestro en la que el atacante se hace pasar por un *host* de confianza para ocultar su identidad, para lograr secuestrar los navegadores o tener acceso a una red.

El suplantador obtiene la dirección IP de un *host* legítimo y procede a alterar las cabeceras de los paquetes TCP/IP, de modo que parezca que los paquetes provienen de ese *host*, y así enviar paquetes desde esa dirección a varios destinatarios de la red. Esto se logra generalmente con programas destinados para este uso, que pueden ser utilizados en cualquier protocolo dentro del *stack* de protocolos de la arquitectura TCP/IP tales como ICMP, UDP o TCP. Generalmente se trabaja en conjunto con otros tipos de suplantación como los que se mencionarán posteriormente.

Un ejemplo de falsificación IP típico es cuando se envía un mensaje ICMP falsificado, la respuesta será recibida por el *hos*t al que pertenece la IP legítima.

Para poder realizar un `IP_SPOOFING` en sesiones TCP, se tienen que tomar en cuenta el comportamiento de dicho protocolo en el envío de paquetes SYN y ACK, teniendo en consideración que el usuario legal de la IP podría cortar la conexión en cualquier momento al recibir paquetes sin haberlos solicitado. Otro aspecto importante que se tiene que tomar en cuenta es que los routers actuales no admiten el envío de paquetes con IP origen no perteneciente a una de las redes que administra (los paquetes falsificados no sobrepasarán el router).

La suplantación de direcciones IP también se puede utilizar para omitir la autenticación basada en direcciones IP. Este proceso puede ser muy difícil sobre todo cuando las relaciones de confianza están entre máquinas de una red y sistemas internos.

En la Figura 1.6 Ataque IP Spoofing Figura 1.6 se muestra el funcionamiento del ataque.

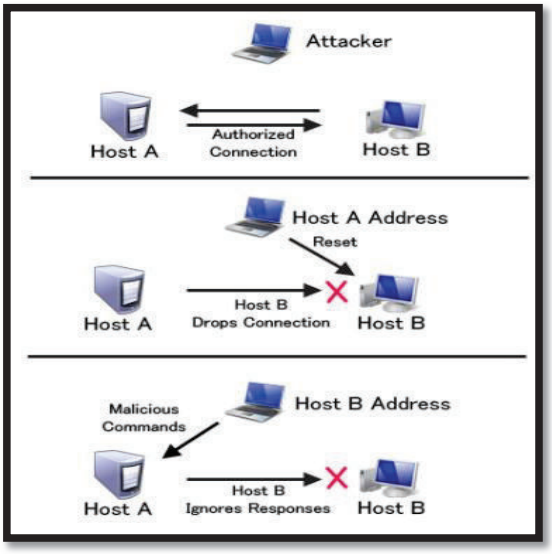

Figura 1.6 Ataque IP Spoofing [19]

1.4.2.2 ARP SPOOFING

Es un conocido ataque de suplantación de identidad que se suele utilizar con fines de investigación o fraudulenta.

El protocolo ARP (*address resolution protocol*) es el encargado de traducir una dirección IP a una dirección MAC que es una dirección de hardware para poder distinguir el equipo destino de una comunicación dentro de una red, es decir asocia una dirección lógica (dirección IP) con una dirección física (dirección MAC).

El ataque se logra creando tramas de solicitudes y respuestas ARP falseadas, de tal manera que se obliga a una máquina de una red de área local a que envíe los paquetes a un *host* atacante en vez de que se envía a su destino legítimo como se muestra en la Figura 1.7.

A pesar de que se trata de una idea muy simple, puede tener un gran impacto negativo dentro de una red, incluyendo denegaciones de servicio, ataques de M*AN-IN-THE-MIDDLE*[41] e inclusive intercepción de datos de protocolos cifrados [20].

Este tipo de ataque es un tanto complejo y además se genera en un solo sentido.

[41] *MAN-IN-THE-MIDDLE* es un tipo de amenaza, donde un agente intercepta una conversación entre dos equipos.

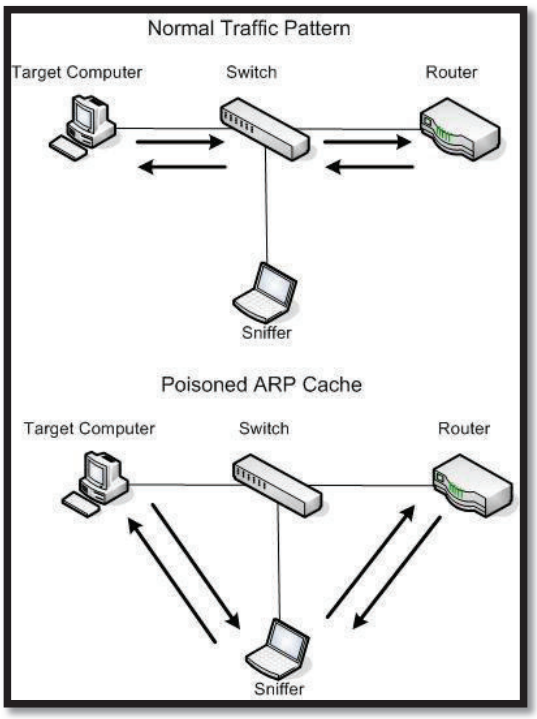

Figura 1.7 Forma en que opera el ataque ARP SPOOFING [21]

1.4.2.3 DNS SPOOFING

Como hay tantos nombres de equipos que a su vez se traducen a su correspondiente dirección IP, los usuarios no pueden manejar toda esa información, es por eso que se utiliza un servidor DNS[42] que maneje de forma centralizada dicha traducción, aprovechando esta dependencia es que se efectúa este ataque.

Este ataque consiste en que un tercer equipo malicioso realiza dicha traducción sin que el usuario note el cambio, como se puede ver en la Figura 1.8, en otras palabras suplanta la identidad del servidor DNS [22].

Una de las muchas razones para las cuales se puede efectuar este ataque es para implementar sitios maliciosos, ya sea para recolectar información de los usuarios como información bancaria o claves de acceso. Otro posible motivo es la utilización

[42] DNS (*Domain Name Service*) es el servicio encargado de resolver las direcciones IP en los respectivos nombres de dominio.

de *widgets*[43] que generalmente piden la autorización para poder instalar el *plug-in*[44] necesario, y de esta forma un atacante puede tener acceso al equipo.

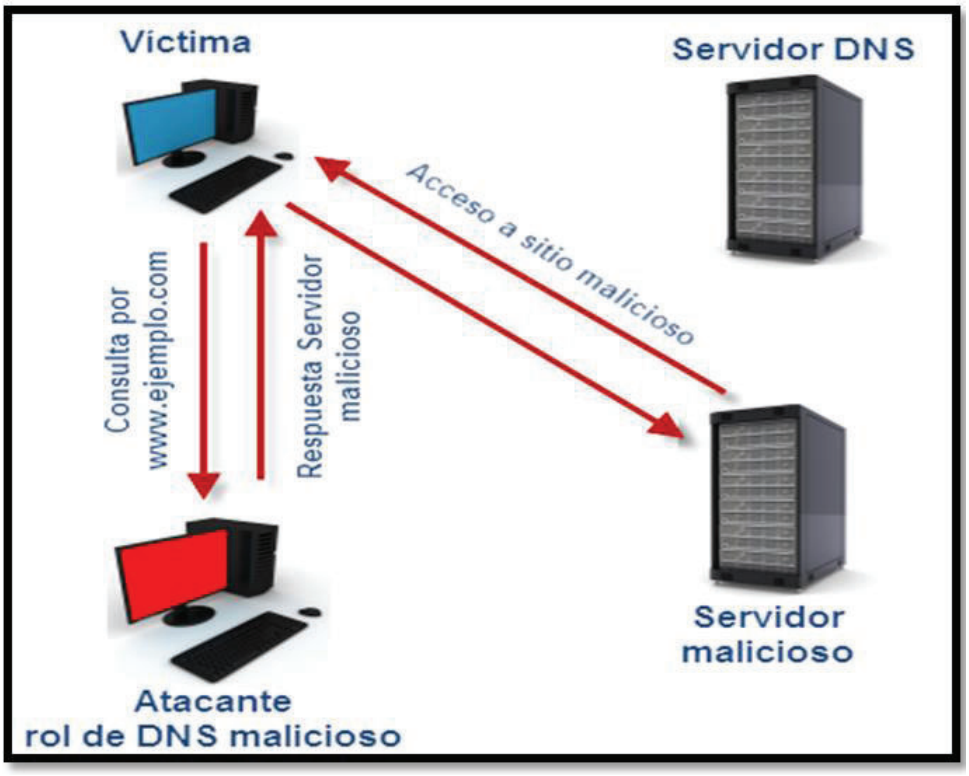

Figura 1.8 Suplantación del servidor DNS

La forma en la que se realiza el ataque es generalmente accediendo ya sea al host de la víctima o al equipo que genera la IP (servidor DHCP[45]) a la cual apunta al servidor DNS. Cambiando dicha IP y en lugar de que apunte al servidor, re-direccione al equipo del atacante.

[43] *Widget* es una pequeña aplicación que se utiliza en las diferentes páginas web, sobre todo para dar servicios multimedia.
[44] *Plug-in* es un complemento que se relaciona con una aplicación para aportarle una función nueva y generalmente muy específica.
[45] Un servidor DHCP (*Dynamic Host Configuration Protocol*) es un protocolo de red que permite a los clientes de una red IP obtener sus parámetros de configuración automáticamente.

CAPÍTULO II

2 ESTRUCTURA GENERAL DE LA RED

Una vez que se han aclarado ciertos conceptos teóricos necesarios para entender tanto el funcionamiento de una *honeynet* como el de la tecnología SDN, se presentará en este capítulo, el diagrama general de la red que se utilizará en este proyecto, en el que constan la estructura y cada uno de los elementos que la conforman, para luego proceder con un análisis particular, explicando el esquema físico y virtual que se usará en cada uno de los ataques.

Después de entender la función de cada equipo tanto físico como virtual en los diferentes escenarios, se procederá a explicar la preparación de cada equipo, indicando las herramientas y configuraciones utilizadas, finalmente, se menciona de forma detallada cómo se genera cada uno de los ataques.

2.1 ELEMENTOS

La topología que se va a utilizar se presenta en la Figura 2.1.

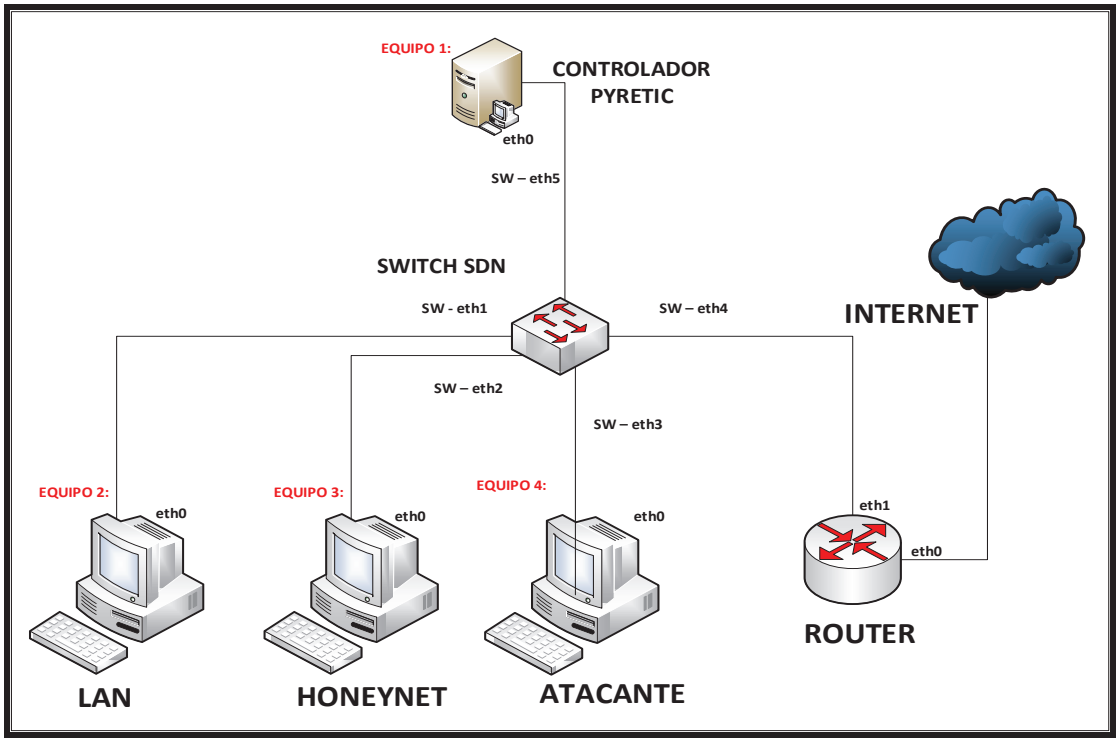

Figura 2.1 Diagrama general de red

2.1.1 EQUIPO 1

El equipo 1, según el diagrama de la Figura 2.1, es el dispositivo donde se ejecutará la aplicación de software que determinará el funcionamiento de la red, dicho equipo será denominado como “Controlador”, para este proyecto la aplicación de software que se usará es Pyretic.

La interfaz eth0 del equipo 1 se conectará mediante cable UTP a cualquier puerto que no tenga habilitado el protocolo OpenFlow del switch SDN.

2.1.1.1 Características

El equipo 1, posee las siguientes características que se muestran en la Tabla 2.1.

EQUIPO 1	
Fabricante	HP
Modelo	ProLiant DL320e Gen8 v2
Procesador	4 CPU x 3,092 GHz
Tipo de Procesador	Intel(R) Xeon(R) CPU E3-1220 v3 @ 3.10GHz
Memoria	7,84 GB
Numero de Interfaces	2
Máquinas Virtuales	3
Almacenamiento:	
Disco Duro	458.25 GB
Red LAN:	
Dirección IP principal	192.168.0.4
Sistema Operativo:	
Sistema operativo	VMware ESXi 5.5.0 Update 2

Tabla 2.1 Características del Equipo 1

2.1.2 EQUIPO 2

El equipo 2, es el dispositivo en donde se va a implementar virtualmente todos los dispositivos de red que forman la verdadera LAN. Entre los dispositivos que se ejecutarán en este equipo se tiene a los *hosts*, el servidor HTTPS, etc.

2.1.2.1 Características

El equipo 2 posee las características que se mencionan en la Tabla 2.2.

EQUIPO 2	
Tipo de ordenador	Equipo basado en ACPI x86
Sistema operativo	Windows 7 Professional Media Center Edition SP1
Placa base:	
Tipo de procesador	DualCore Intel Core 2 Duo E6750, 2666 MHz (8 x 333)
Chipset de la Placa Base	Intel Eaglelake
Memoria del Sistema	2009 MB
Tipo de BIOS	AMI (12/23/09)
Video:	
Tarjeta gráfica	Intel(R) G41 Express Chipset (Microsoft Corporation - WDDM 1.1)
Tarjeta de sonido	
Audio:	Realtek ALC662 @ Intel 82801GB ICH7 - High Definition Audio Controller [A-1]
Controlador IDE	Controladora de almacenamiento ATA serie Intel(R) 82801GB/GR/GH (familia ICH7) - 27C0
Disco duro	MAXTOR STM3250310AS ATA Device (250 GB, 7200 RPM, SATA-II)
Red	
Dirección IP principal	192.168.0.2
Dirección MAC principal	00-00-00-00-00-02 (user-defined)
Tarjeta de Red	NIC Gigabit Ethernet PCI-E de la familia Realtek RTL8168C(P)/8111C(P)

Tabla 2.2. Características del Equipo 2

2.1.3 EQUIPO 3

El equipo 3, es el dispositivo en donde se va a implementar virtualmente todos los dispositivos de red que formar la *honeynet*. Entre los dispositivos que se ejecutarán en este equipo se tiene a los *hosts*, el servidor de HTTPS y demás equipos copia.

2.1.3.1 Características

El equipo 3 posee las características que se mencionas en la Tabla 2.3.

EQUIPO 3

Tipo de ordenador	Equipo basado en ACPI x64
Sistema operativo	Windows 8.1 Professional
Placa base:	
Tipo de procesador	DualCore Intel Pentium D 915, 2800 MHz (14 x 200)
Chipset de la Placa Base	Intel Broadwater i946GZ
Memoria del Sistema	3061 MB (DDR2 SDRAM)
Tipo de BIOS	Intel (09/11/06)
Monitor:	
Tarjeta gráfica	Intel(R) 946GZ Express Chipset Family (Microsoft Corporation - WDDM 1.1) (384 MB)
Multimedia:	
Tarjeta de sonido	SigmaTel STAC9227X @ Intel 82801GB ICH7 - High Definition Audio Controller [A-1]
Almacenamiento:	
Controlador IDE	Controladora de almacenamiento ATA serie Intel(R) 82801GB/GR/GH (familia ICH7) - 27C0
Disco duro	SAMSUNG SP2504C ATA Device (250 GB, 7200 RPM, SATA-II)
Red:	
Dirección IP principal	192.168.0.2
Dirección MAC principal	00-00-00-00-00-02 (user-defined)
Tarjeta de Red	Conexión de red Intel(R) PRO/100 VE (192.168.0.8)

Tabla 2.3. Características del Equipo 3

2.1.4 EQUIPO 4

El equipo 4, es el dispositivo que actuará como el atacante, en este se instalará la distribución Kali Linux como sistema operativo.

2.1.4.1 Características

El equipo 4 posee las características que se mencionan en la Tabla 2.4.

EQUIPO 4	
Tipo de ordenador	Equipo basado en ACPI x64

Sistema operativo	Linux 3.12-kali1-amd64 (x86_64)
Placa base:	
Tipo de procesador	8x Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
Chipset de la Placa Base	Intel Broadwater i946GZ
Memoria del Sistema	8063MB (1143MB used)
Tipo de BIOS	Intel (09/11/06)
Monitor:	
Tarjeta gráfica	Mesa DRI Intel(R) Ivybridge Desktop
Multimedia:	
Tarjeta de sonido	HDA-Intel - HDA Intel PCH
Almacenamiento:	
Controlador IDE	Controladora de almacenamiento ATA serie Intel(R) 82801GB/GR/GH (familia ICH7) - 27C0
Disco duro	SAMSUNG SP2504C ATA Device (250 GB, 7200 RPM, SATA-II)
Red:	
Dirección IP principal	192.168.0.3
Dirección MAC principal	00-00-00-00-00-03 (user-defined)
Tarjeta de Red	Conexión de red Intel(R) PRO/100 VE (192.168.0.3)

Tabla 2.4. Características del Equipo 4

2.1.5 SWITCH SDN

Se lo puede considerar como el equipo central de la red, ya que todos los demás equipos se conectan a este, cada paquete que llega es encapsulado añadiendo una cabecera OpenFlow y se lo envía al controlador como un `packet IN`, una vez que el controlador lo procesa lo regresa al switch como un `packet OUT`, el switch lo desencapsula y lo trata de acuerdo a la política que el controlador haya definido para este [23].

2.1.5.1 Características

El switch posee las características que se mencionan en la Tabla 2.5.

SWITCH SDN

Modelo	HP 3500-24G-PoE+ yl
Interfaz LAN	20 puertos 10/100/1000 , Soporta máximo 4 puertos 10Gb
Latencia	1000 Mbps < 3.4 us, 10 Gbps < 2.1 us
Throughput	Sobre los 75.7 Mpps
Capacidad de Ruteo y Conmutación	101.8 Gbps
Velocidad de procesamiento	105.6 Gbps
Capacidad puertos PoE	398 W
Administración	Centro de administración Inteligente (IMC), interfaz de línea de comandos, explorador web, menú de configuración, administración puerto serial RS-232C
Memoria:	
Flash	4 MB
Compact Flash	128 MB
DDR SDRAM	256 MB
Red LAN:	
Dirección IP VLAN 1	192.168.0.250

Tabla 2.5 Características del Switch SDN

2.1.6 ROUTER

Este dispositivo poseerá varias funciones entre las que se incluyen: permitir la conexión a Internet, actuar como servidor DHCP, servidor DNS y *gateway*. [24] [25]

2.1.6.1 Características

Las características físicas del router se mencionan en la Tabla 2.6.

ROUTER	
Modelo	D-Link DI-624
Interfaz WAN	RJ-45 10BASE-T/100BASE-TX
Interfaz LAN	802.11g Wireless (54 Mbps) 4 RJ-45 10BASE-T/100BASE-TX

Funciones de Gateway	Translación de direcciones de Red (NAT) Servidor DHCP
Configuración y Administración	Web
Rango de frecuencias	2.4 – 2.4835 GHz
Antena	Ganancia 2 dBi, antena desmontable, conector SMA inverso.
Red WAN:	
Dirección IP principal	172.31.9.72
Dirección MAC principal	00-13-46-cf-08-85
Red LAN:	
Dirección IP principal	192.168.0.1
Dirección MAC principal	00-13-46-cf-08-84

Tabla 2.6 Características del Router

2.2 TOPOLOGÍAS ESPECÍFICAS DE ACUERDO AL ATAQUE

Como se ha mencionado anteriormente, la muestra el diagrama general de la topología de la red a implementar en este proyecto, el switch SDN ha sido configurado para que sus puertos del 1 al 4 trabajen como puertos OpenFlow, los demás puertos podrán trabajar como puertos normales, uno de ellos, el puerto 5 del switch, servirá para comunicar al switch SDN con el controlador que se encuentra virtualizado en el equipo 1, de esta forma podrán establecer y mantener la conexión.

Las características del controlador se especifican a continuación en la Tabla 2.7.

CONTROLADOR	
Tipo de ordenador	Equipo basado en ACPI x86
Sistema operativo	Ubuntu 14.04.2 LTS
Placa base:	
Tipo de procesador	CPU E3-1220 v3 @ 3.10GHz
Chipset de la Placa Base	4x Intel(R) Xeon(R)

Memoria del Sistema	3083MB
Tipo de BIOS	AMI (07/30/2013)
Tarjeta de sonido	
Almacenamiento:	
Controlador IDE	ATA VMware Virtual I
Disco duro	NECVMWar VMware IDE CDR10
Red	
Dirección IP principal	192.168.0.100
Tarjeta de Red	VMware VMXNET3 Ethernet Controller (rev 01)

Tabla 2.7 Características del controlador

En el puerto 1 del switch se conectará el equipo 2, el cual representará para cada uno de los ataques la infraestructura de LAN, en dicho equipo se virtualizan *hosts*, servidores u otros dispositivos que se requiera.

En el puerto 2 del switch se conectará el equipo 3, el cual representará para los diferentes ataques a la *honeynet*, este al igual que el equipo 2, virtualizá los dispositivos necesarios en cada ataque, dichos equipos serán una copia exacta del implementado en la LAN, incluyendo sus mismas direcciones IP y mismas direcciones MAC.

En el puerto 3 se conectará el equipo 4 que es el equipo desde el cual se realizarán los diferentes ataques, este equipo representa a un *host* interno de la LAN a pesar de encontrarse en un puerto diferente.

En el último puerto, el número 4, se conectará al *gateway*, de esta forma se tendrá acceso a Internet desde cualquier equipo que está en la LAN o en la *honeynet* si se requiere y además podría representar el camino por el cual puede llegar un ataque desde internet.

A continuación se muestra cada una de las topologías que se utilizarán para los distintos ataques y se explica el papel que cumple cada dispositivo en la red.

2.2.1 ATAQUE ARP SPOOFING

Para el desarrollo de este ataque, se procederá a trabajar con una topología como la que se muestra en la Figura 2.2, en la cual se observa que el ataque proviene del equipo 4 o desde Internet.

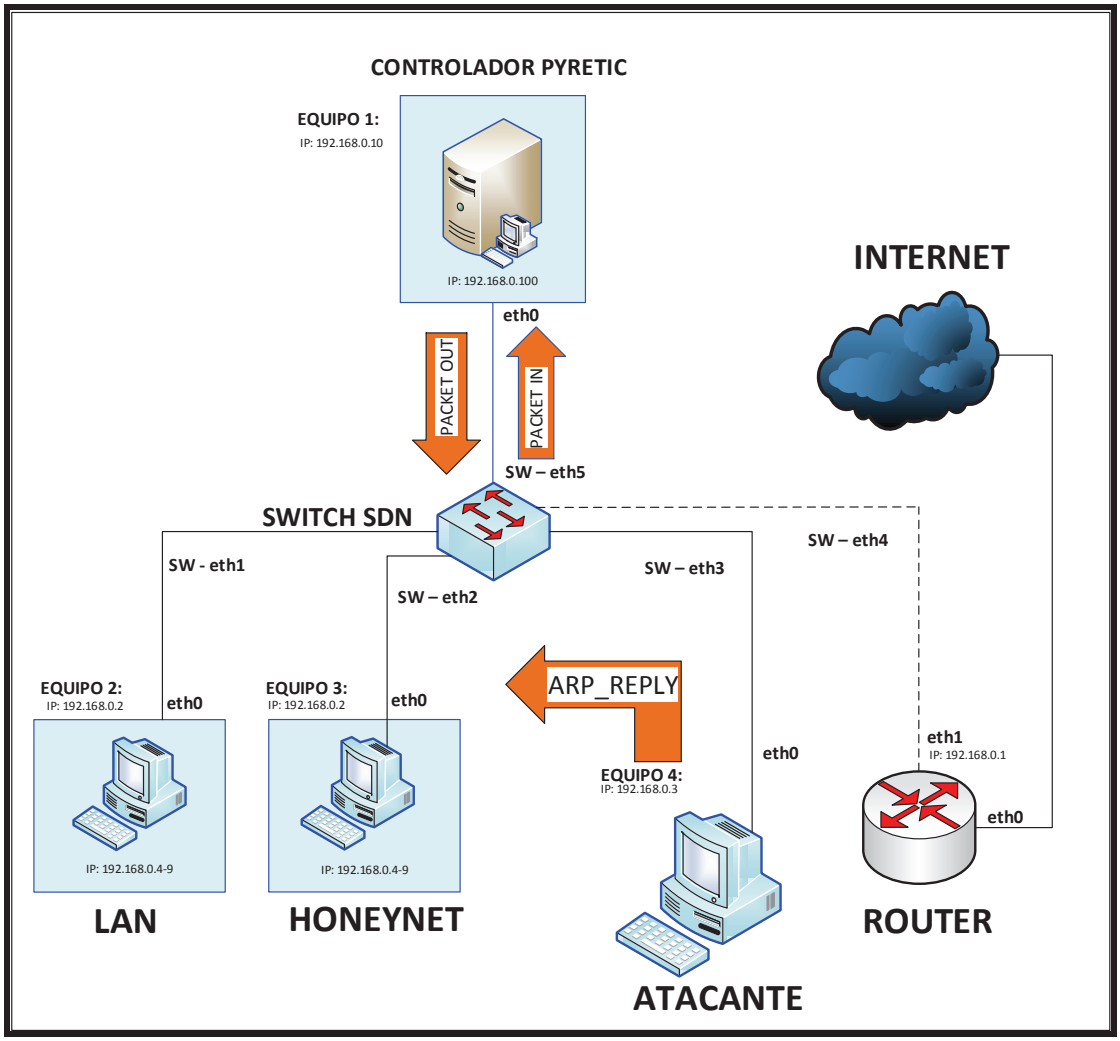

Figura 2.2 Diagrama del ataque `ARP_SPOOFING`

El switch SDN envía cada uno de los paquetes al controlador en forma de `PACKET IN`, el controlador lo procesa mediante el módulo que permita detectar si se trata de un ataque `ARP_SPOOFING` y luego lo devuelve al switch como un `PACKET OUT`, y a su vez, el controlador le indicará si debe enviarlo hacia la LAN o desviarlo a la *honeynet*.

Una vez que las tablas ARP estén pobladas con la información de los diferentes equipos de la red, el atacante puede generar ARP SPOOFING, para lo cual enviará continuamente paquetes del tipo ARP_REPLY de tal forma que la dirección IP de la víctima se asocie a la dirección MAC del atacante, en la cual todos los paquetes que se dirijan hacia la dirección IP de la víctima, se los enviará hacia el atacante y no al equipo de la víctima.

Es fundamental que la dirección IP del atacante se encuentre en la tabla ARP de la víctima.

2.2.2 ATAQUE IP SPOOFING

El ataque proviene del equipo 4 o desde Internet, y llega al switch SDN como se muestra en la Figura 2.3. El switch envía cada uno de los paquetes al controlador en forma de PACKET IN, este procesa el módulo que permita detectar si se trata de un ataque de tipo IP_SPOOFING y luego lo devuelve al switch como un PACKET OUT, a su vez, el controlador le indicará si debe enviarlo hacia la LAN o desviarlo a la *honeynet*. Al tratarse de una suplantación de la dirección IP, el controlador deberá discriminar entre usuarios legítimos y atacantes mediante el uso de direcciones MAC, cuando reciba un paquete proveniente de una dirección MAC conocida y cuya IP ha sido modificada, el controlador debe descubrir si el cambio de dirección IP fue un cambio válido, realizado ya sea manualmente, por DHCP o que en realidad se trata de un ataque. El controlador generará un paquete ECHO_REQUEST dirigido hacia la víctima para comprobar si dos equipos responden a dicha petición, de ser el caso se confirma que se trata de un ataque ya que en la LAN se encuentran dos equipos con la misma dirección IP, al saber que es un atacante se le direcciona a la *honeynet*.

Una vez que direcciona el tráfico a la *honeynet* se intentará monitorear cualquier movimiento que realice.

Debido a que la *honeynet* y la LAN no tienen conectividad entre sí, no hay ninguna posibilidad de que el atacante acceda a la LAN a través de la *honeynet* una vez se encuentre en ella.

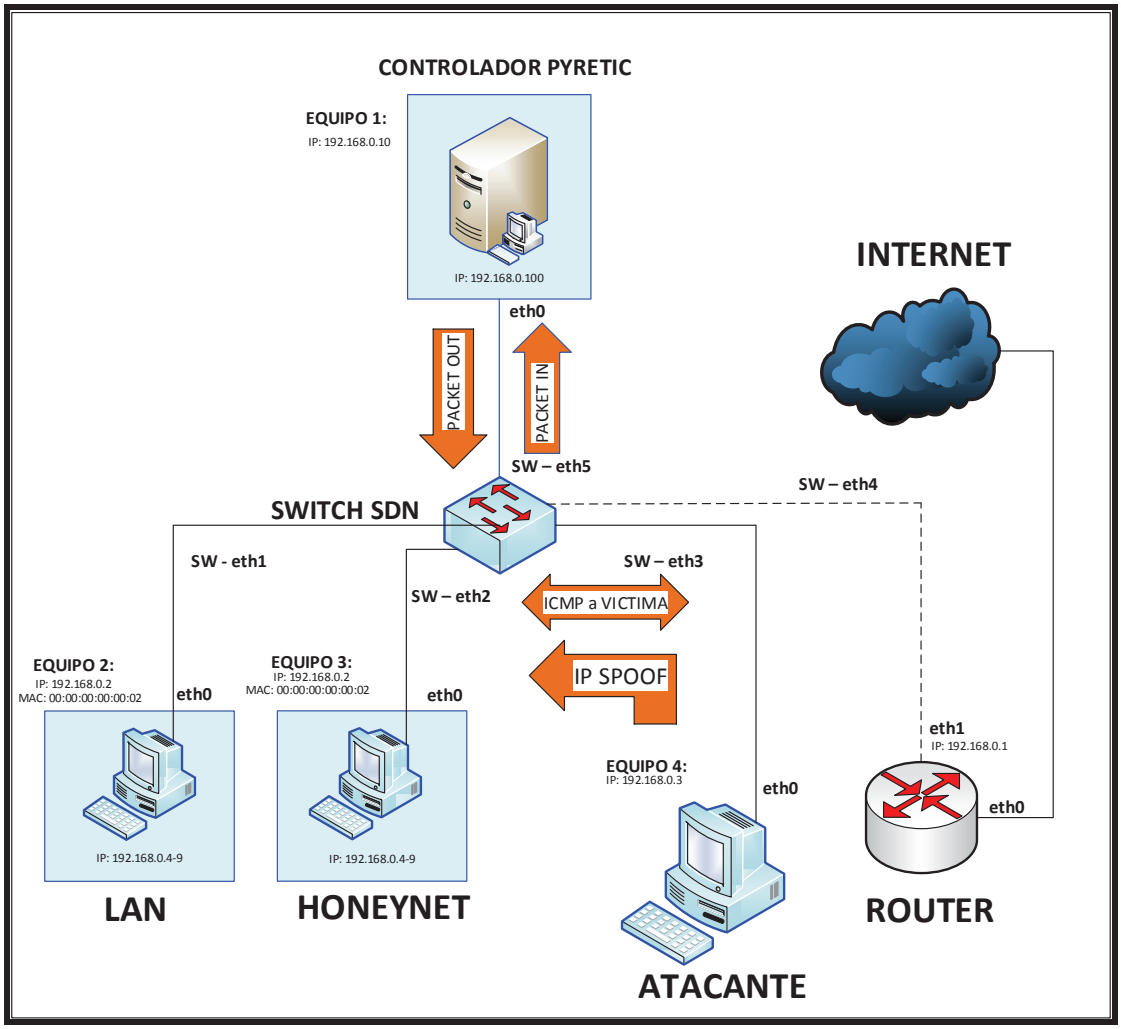

Figura 2.3 Diagrama de red para el ataque `IP_SPOOFING`

2.2.3 TOPOLOGÍA PARA EL ATAQUE TCP SYN FLOOD

El ataque proviene del equipo 4 o desde Internet, llega al switch SDN, el mismo que envía cada uno de los paquetes al controlador en forma de PACKET IN como se muestra en la Figura 2.4, este procesa el modulo que permita detectar si se trata de un ataque TCP_SYN_FLOOD y luego lo devuelve al switch como un PACKET OUT, a su vez, el controlador le indicará si debe enviarlo hacia la LAN o desviarlo a la *honeynet*.

Este ataque trabaja con el *handshake* de TCP como ya se mencionó anteriormente, el atacante inunda la red con paquetes `TCP_SYN` de intento de conexión, el

controlador debe determinar si se trata de un ataque y entonces proceder a desviar todo el tráfico que se genera desde la dirección IP del atacante hacia la *honeynet*.

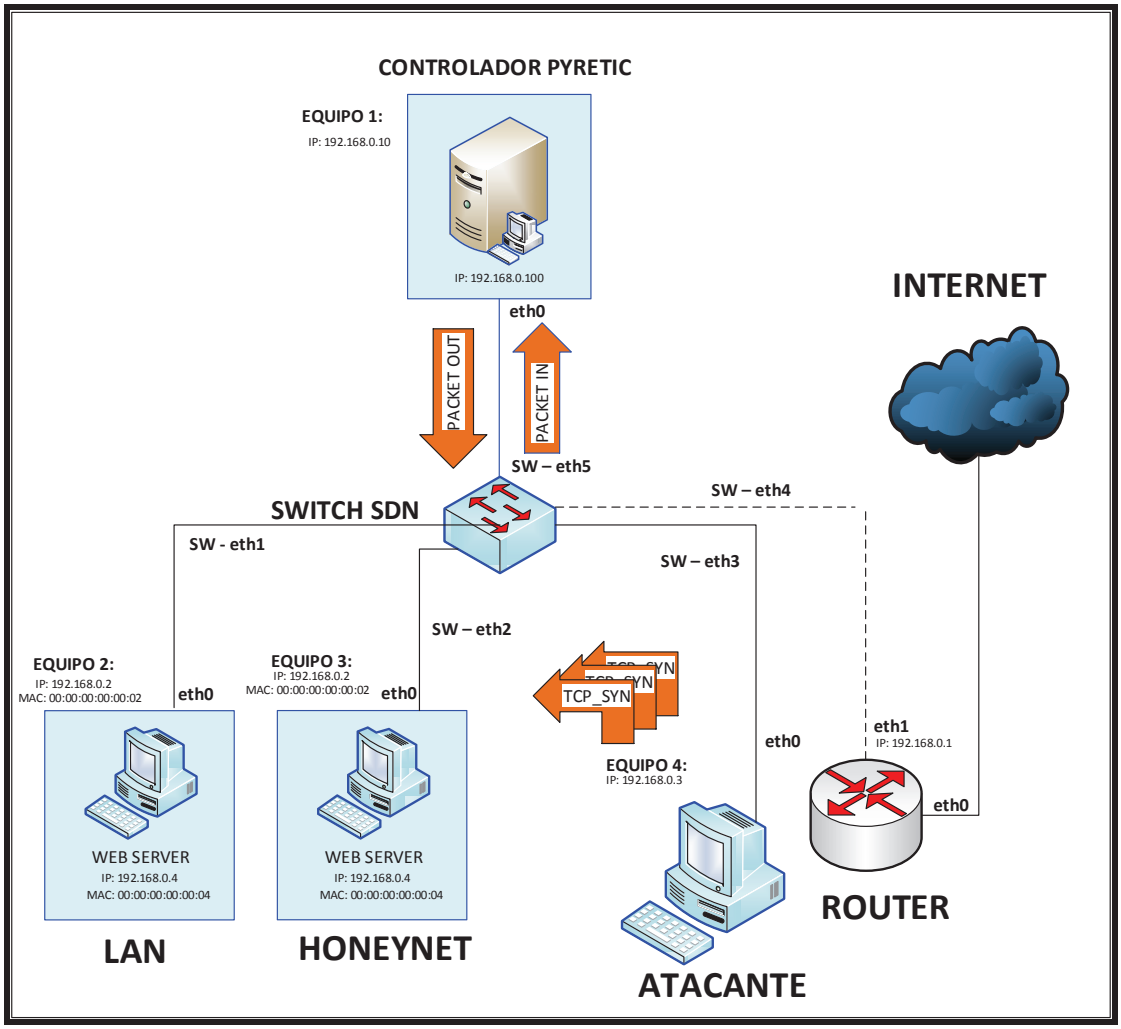

Figura 2.4 Diagrama de red para el ataque `TCP_SYN_FLOOD`

Si el equipo cuya dirección IP desde la cual se realizó el ataque se comporta como un usuario legítimo, se le devolverá la conectividad con la LAN.

Para este ataque se requiere un servidor web que será virtualizado tanto en el equipo 2 como en el 3.

2.2.3.1 Características del servidor web

El equipo virtual ejecuta el servidor web, dispone de las características que se mencionan en la Tabla 2.8.

SERVIDOR WEB	
Tipo de ordenador	Equipo multiprocesador ACPI
Sistema operativo	Windows XP Professional
Service Pack	Service Pack 3
Placa base:	
Tipo de procesador	DualCore Intel Pentium D, 2800 MHz
Chipset de la Placa Base	Intel 82440BX/ZX
Memoria del Sistema	1024 MB
Tipo de BIOS	AMI (12/23/09)
Video:	
Tarjeta gráfica	VMware SVGA II
Tarjeta de sonido:	
Audio:	Realtek ALC662 @ Intel 82801GB ICH7 - High Definition Audio Controller [A-1]
Controlador IDE	Controladora IDE principal de bus PCI Intel(R) 82371AB/EB
Disco duro	VMware Virtual IDE Hard Drive (14 GB, IDE)
Red:	
Dirección IP principal	192.168.0.5
Dirección MAC principal	00-00-00-00-00-05 (user-defined)
Tarjeta de Red	VMware Accelerated AMD PCNet Adapter (192.168.0.5)

Tabla 2.8 Características del servidor WEB

2.2.4 ATAQUE THC SSL DoS

El ataque proviene del equipo 4 o desde Internet, llega al switch SDN, el mismo que envía cada uno de los paquetes al controlador en forma de PACKET IN, este procesa el modulo que permita detectar si se trata de un ataque THC_SSL_DoS y luego lo devuelve al switch como un PACKET OUT, a su vez, el controlador le indicará si debe enviarlo hacia la LAN o desviarlo a la *honeynet.*

Este ataque funciona en HTTPS y a su vez sobre TCP. Una vez que el ataque a la LAN comienza, le llega el establecimiento de conexión TCP una vez concluido empieza el intercambio de credenciales SSL. El ataque se basa en el intercambio de credenciales ya que el atacante nunca acepta las credenciales continuando la renegociación, obligando al servidor a consumir una gran cantidad de recursos

durante el transcurso, esto se debe a que el servidor realiza mucho mayor procesamiento que un cliente durante la renegociación, una vez que se detecta este comportamiento el controlador direcciona a todo paquete que llega de este la dirección IP del atacante a la *honeynet*.

Una vez que se envía a la *honeynet* ya está en la renegociación de SSL, la réplica de la víctima al no saber por qué le llega estos paquetes se perderá la comunicación momentáneamente mientras se establezca la conexión TCP y continuará el ataque renegociando los parámetros SSL, indefinidamente.

En el diagrama de la Figura 2.5 se muestra la topología a utilizar para efecto de pruebas, de los equipos 2 y 3 se procederá a utilizar el servidor web y su réplica anteriormente mencionada, y el ataque se realizará desde al equipo 4.

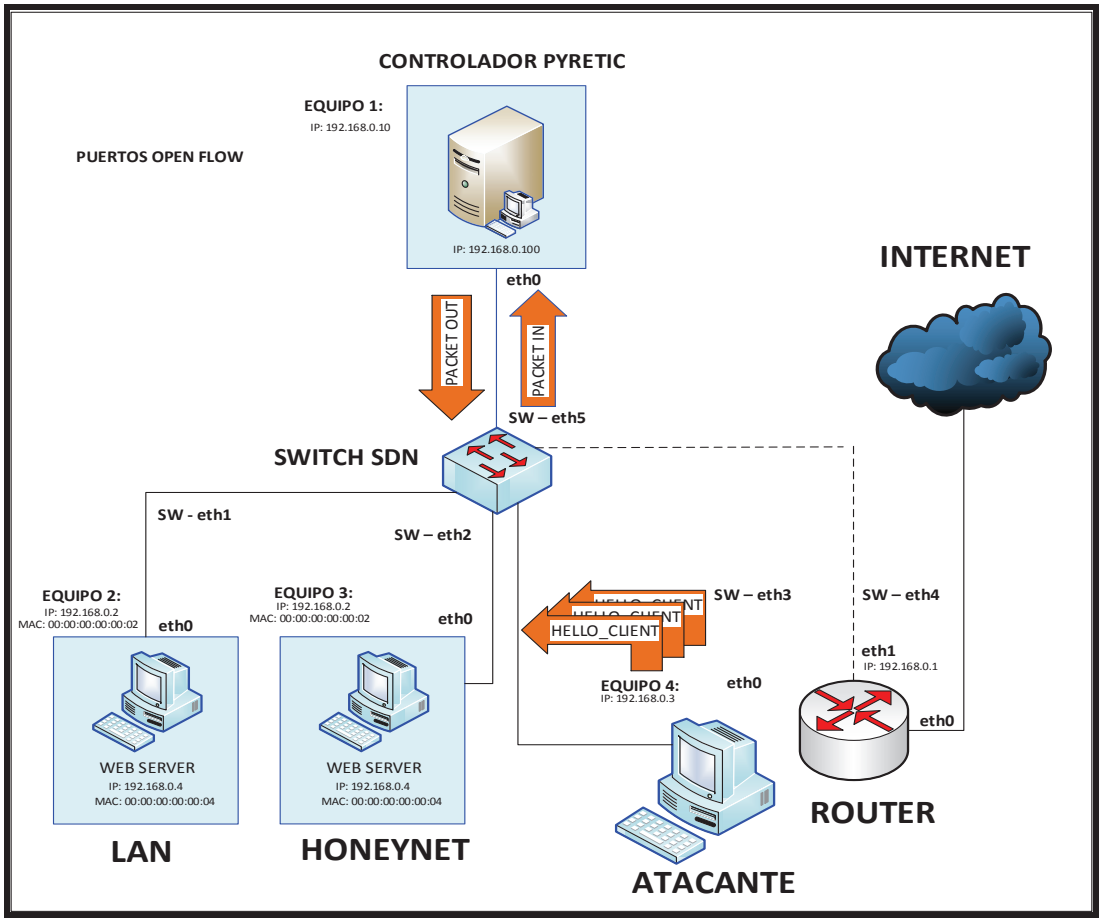

Figura 2.5 Diagrama de red para el ataque `THC_SSL_DoS`

2.2.5 ATAQUE SMURF

El ataque SMURF se realizará utilizando la topología que se muestra en la Figura 2.6, en la cual se muestra a los equipos 2 y 3 como los representantes de la LAN y *honeynet* respectivamente y el ataque se efectuará desde el equipo 4.

Para este ataque se debe controlar que todos los paquetes de tipo ICMP no tengan como dirección de destino la dirección IP de *broadcast*, si esto sucede, se tomará como un ataque y dichos paquetes se desviarán a la *honeynet*.

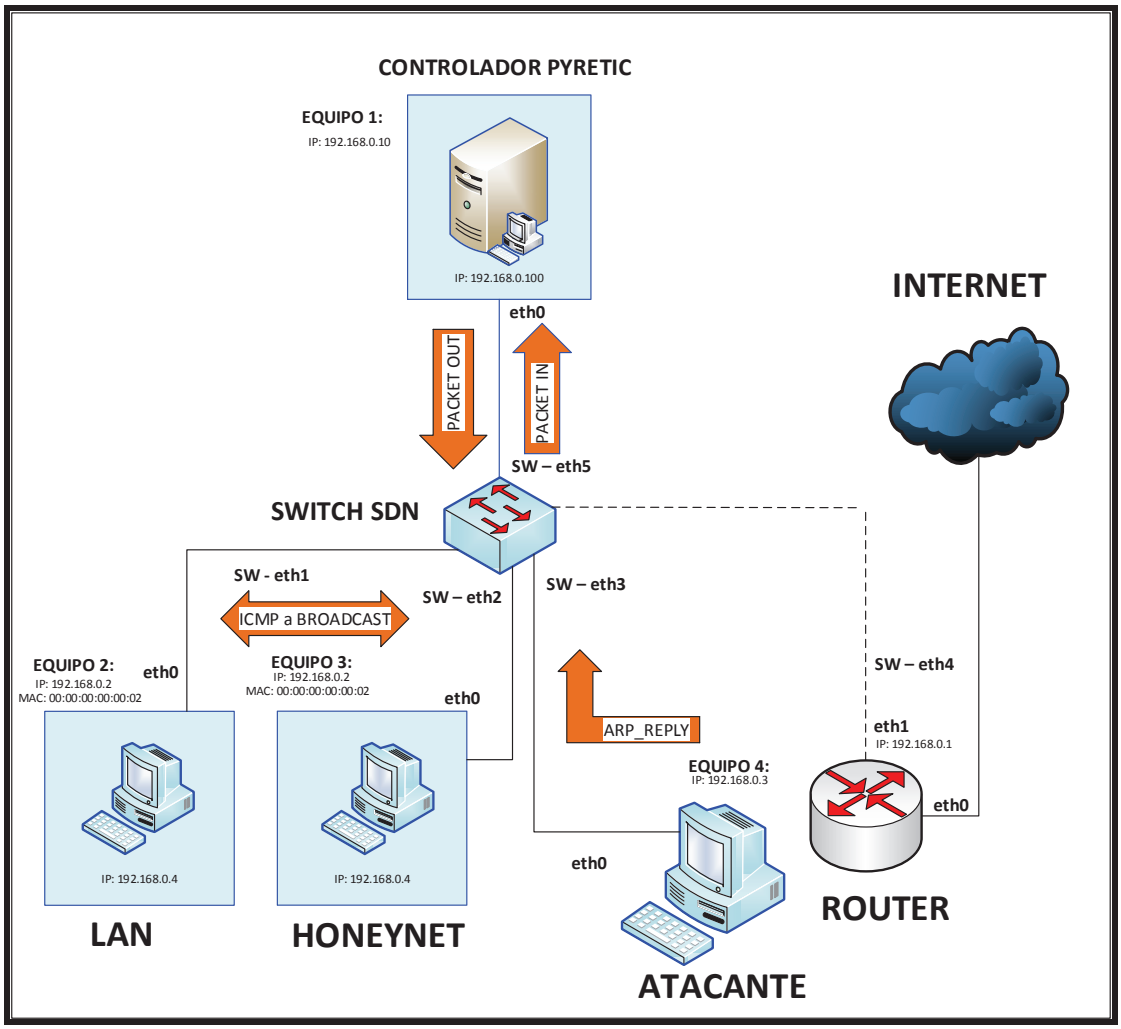

Figura 2.6 Diagrama de red para el ataque SMURF

En la honeynet se realizará el ataque completo puesto que el objetivo no es detener el ataque, y al tratarse de una petición ICMP, todos los equipos responderán a la réplica de la víctima.

2.2.6 ATAQUE DNS SPOOFING

Como se muestra en la Figura 2.7, este ataque utilizará una topología similar a las usadas en los ataques anteriores, en la que los equipos 2 y 3 representan la LAN y la *honeynet* respectivamente, y el ataque se lo realizará desde el equipo 4.

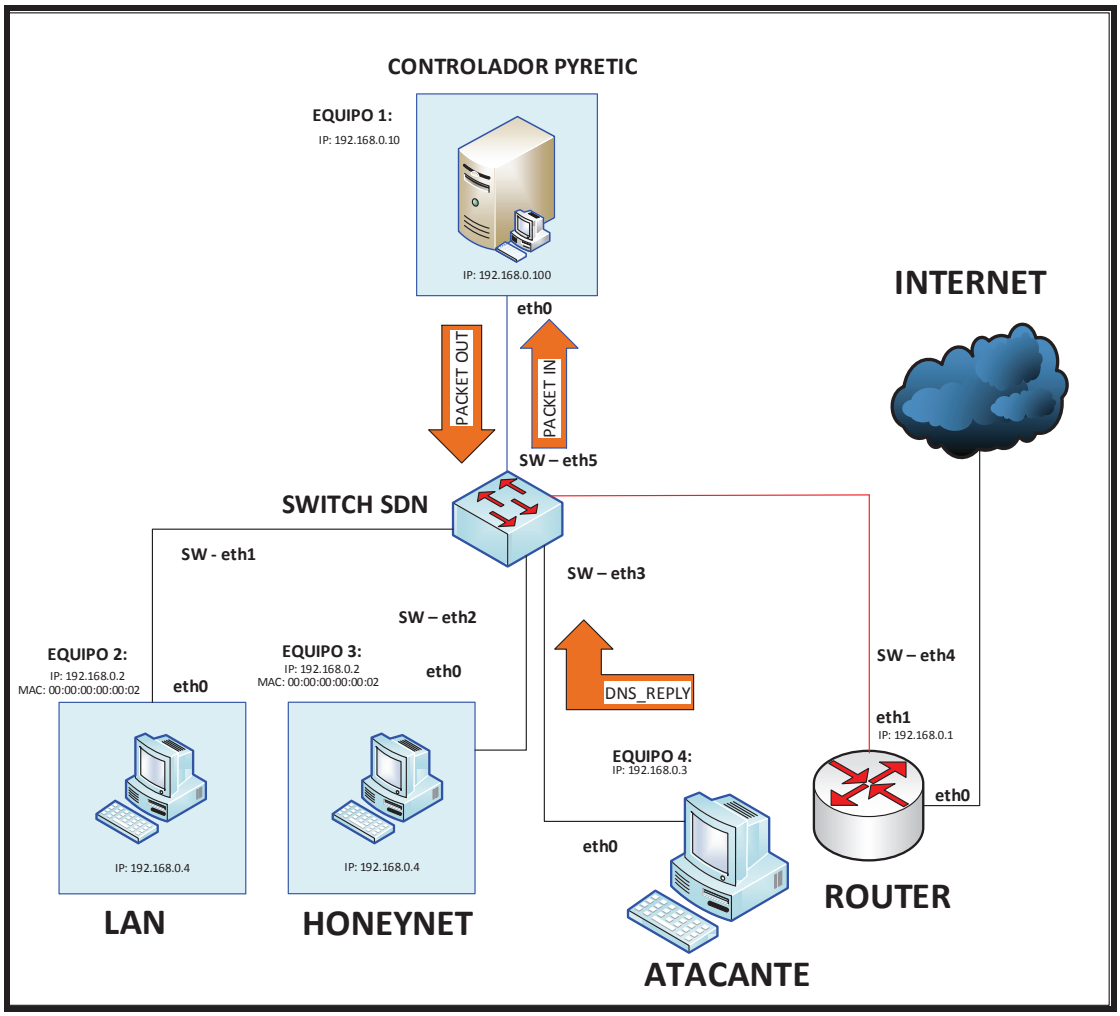

Figura 2.7 Diagrama DNS_SPOOFING

El ataque se produce cuando el atacante empieza a enviar paquetes ARP falsos en los que indica que todas las direcciones IP de la red tienen su dirección MAC. Esto provoca que todos los *hosts* de la red, al momento de querer comunicarse con

cualquier otro equipo, envíen los paquetes al atacante. El atacante actuará como un switch, conmutando los paquetes que considera son poco relevantes, pero cuando recibe paquetes que considera importantes, entonces actúa como el verdadero destinatario, haciendo creer que efectivamente es el equipo con el que deseaba establecer la comunicación.

Cuando se genera una petición DNS desde un equipo de la LAN que tenga como MAC destino una dirección diferente a la dirección del *gateway* o el servidor DNS, se considera que es un ataque por lo tanto la dirección MAC destino del paquete corresponda a la del atacante, entonces cualquier paquete que provenga de esa dirección MAC se desviará a la *honeynet*.

Cabe mencionar que este ataque se lo puede prevenir de la forma mencionada en la sección 2.2.1.

2.3 CONFIGURACIÓN DE EQUIPOS

La configuración de los diferentes equipos en el prototipo desarrollado, abarca tanto la configuración de equipos físicos como de equipos virtuales, e inclusive la instalación y configuración de las distintas aplicaciones necesarias para los diferentes ataques.

2.3.1 CONFIGURACIÓN DEL EQUIPO 1

Como se presentó en la sección 2.2.1, el equipo 1 es el dispositivo en donde se encuentra virtualizado el controlador Pyretic, por lo tanto su configuración involucra instalación y configuración de dicho controlador.

2.3.1.1 Instalación de Pyretic

Para la instalación del controlador Pyretic se debe realizar:

- Descargar de [26], la máquina virtual en formato VMDK[46] que viene configurada con todas las librerías necesarias.
  La arquitectura de la máquina virtual que se utiliza es de 64 bits, ya que el identificador que usa el switch HP que se utiliza en el prototipo tiene una longitud demasiado larga para funcionar adecuadamente en una arquitectura de 32 bits.

[46] VMDK (Virtual Machine Disk) es un formato de archivo utilizado por productos de VMware.

- Usando el archivo VMDK, se debe generar la máquina virtual en VMware Workstation, antes de subirla al servidor, ya que el sistema operativo esxi no puede correr directamente archivos con dicha extensión.
- Una vez que se cargan los archivos relacionados al disco duro de la máquina virtual al servidor entonces se puede crear una nueva máquina y asignarle el disco duro convertido.

2.3.1.2 Instalación del entorno de desarrollo

El entorno seleccionado para desarrollar la aplicación es PyCharm, este entorno permite trabajar en modo depuración, además brinda un ambiente fácil e intuitivo para utilizar [27].

Para su instalación se recomienda realizar lo siguiente:

- El Segmento de código 2.1 presenta los comandos para actualizar el repositorio requerido.

```
wget -q -O - http://archive.getdeb.net/getdeb-archive.key | sudo
apt-key add –
sudo sh -c 'echo "deb http://archive.getdeb.net/ubuntu
$(lsb_release -sc)-getdeb apps" >>
/etc/apt/sources.list.d/getdeb.list
sudo apt-get update
```

Segmento de código 2.1 Actualización de repositorios

- Con el comando presentado en el Segmento de código *2.2* se procede a instalar el programa PyCharm.

```
sudo apt-get install pycharm
```

Segmento de código 2.2 Instalación del entorno de desarrollo

Para ejecutar el programa se emplea el comando indicado en el Segmento de código 2.3.

```
sudo pycharm
```

Segmento de código 2.3 Ejecución del Entorno de Desarrollo

2.3.1.3 Instalaciones adicionales

Para ingresar al controlador se empleará SSH, para lo cual se usará el programa PuTTY, descargado de [28]. Una captura de pantalla del programa se muestra en la Figura 2.8.

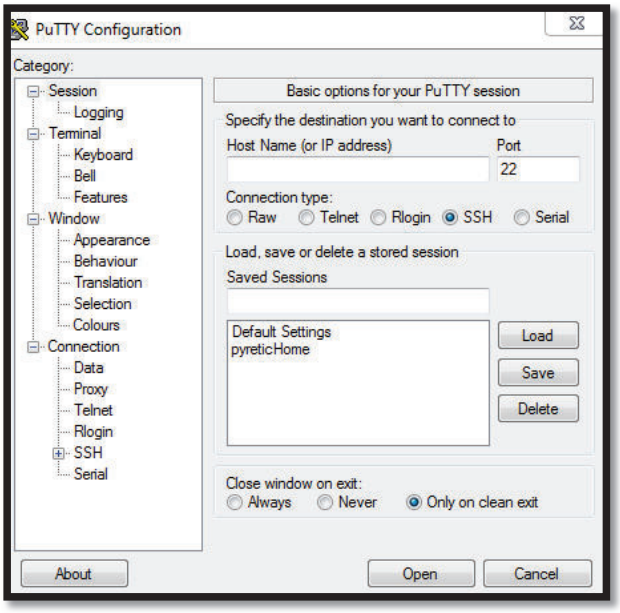

Figura 2.8 Programa PuTTY

Adicionalmente se utiliza el programa Xming X Server para Windows, el cual permite que el controlador pueda presentar sus interfaces gráficas en el equipo Windows, y se lo descarga de [29].

2.3.2 CONFIGURACIÓN DEL EQUIPO 2 Y EQUIPO 3

El equipo 3 tendrá las mismas configuraciones del equipo 2, ya que se comporta como una réplica.

Cada equipo está conformado por un sistema operativo sobre el cual se instalará el programa VMware para virtualizar el sistema operativo Windows XP, en donde se instalara el servidor web con soporte de HTTPS.

2.3.2.1 Configuración de VMware

Debido a que ante el atacante se trata del mismo equipo, se debe configurar la dirección MAC en los dos equipos para que sea la misma, para lo cual se debe seguir la ruta y en la ventana buscar la opción MAC *Address.* En la

Figura 2.9 se muestra una captura de pantalla de la ventana donde se configura la dirección MAC.

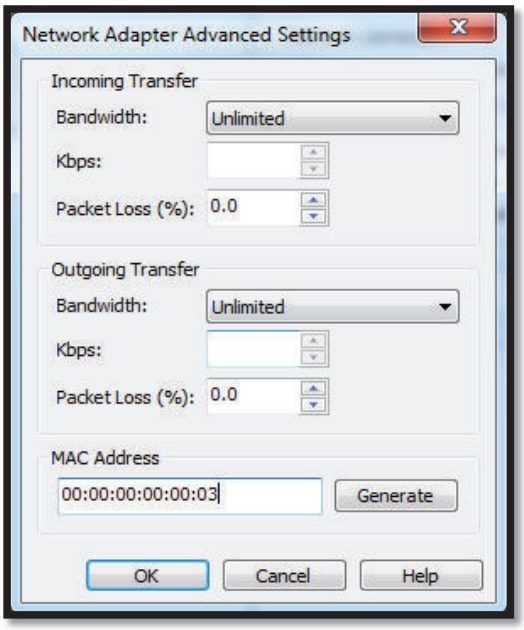

Figura 2.9 Configuración de la dirección MAC en VMware

2.3.2.2 Instalación del Servidor Web

Como ya se mencionó se utiliza Windows XP SP3, ya que la versión de IIS[47] que este incluye soporta la renegociación, fundamental para realizar el ataque THC_SSL_DoS.

Para la instalación de IIS se debe realizar lo siguiente:

[47] IIS (*Internet Information Services)* es el servidor web utilizado por Microsoft Windows.

- Proceder con la activación del servicio IIS, la cual se menciona en la Figura 2.10, siguiendo la ruta `Panel de control\Todos los elementos de Panel de control\Programas y características\Características de Windows.`

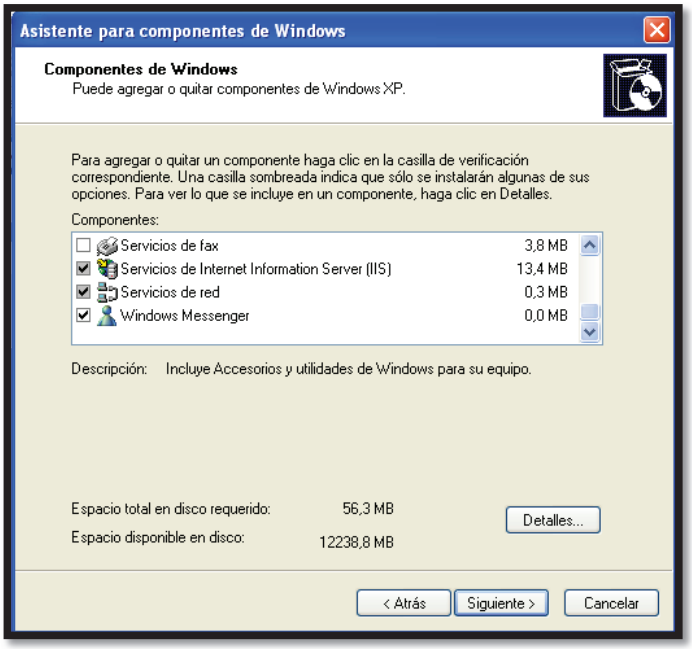

Figura 2.10 Activación del servicio IIS

- Ejecutar el Administrador de Equipos, una captura de este proceso se muestra en la Figura 2.11.

Figura 2.11 Ejecución del Administrador de Equipos

- Una vez en el Administrador de Equipos, se debe buscar en el submenú Servicios de *Internet Information Server*, Sitios Web, y en este se selecciona Sitio Web Predeterminado, como se muestra en la Figura 2.12.

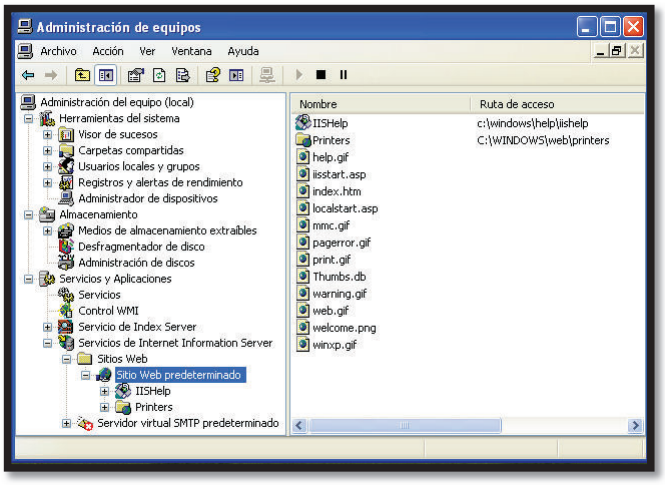

Figura 2.12 Panel de Administrador de Equipos

- Presionar el clic derecho y seleccionar propiedades, se desplegará la ventana como se muestra en la Figura 2.13, en la que se selecciona Certificado de servidor, para generar los certificados requeridos para el protocolo SSL.

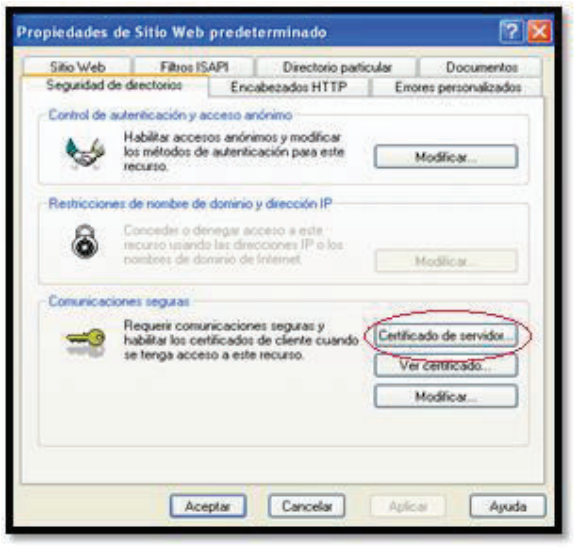

Figura 2.13 Propiedades sitio web predeterminado de IIS

- Una vez abierto el Asistente para certificados de servidor Web, se procede con los pasos que se muestran en la Figura 2.14, Figura 2.15 y Figura 2.16.

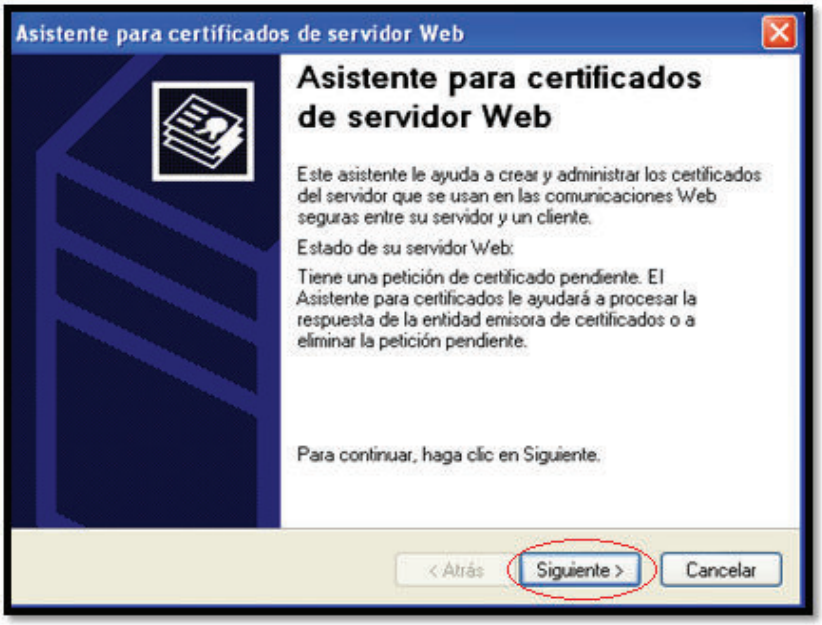

Figura 2.14 Asistente para certificados de servidor Web

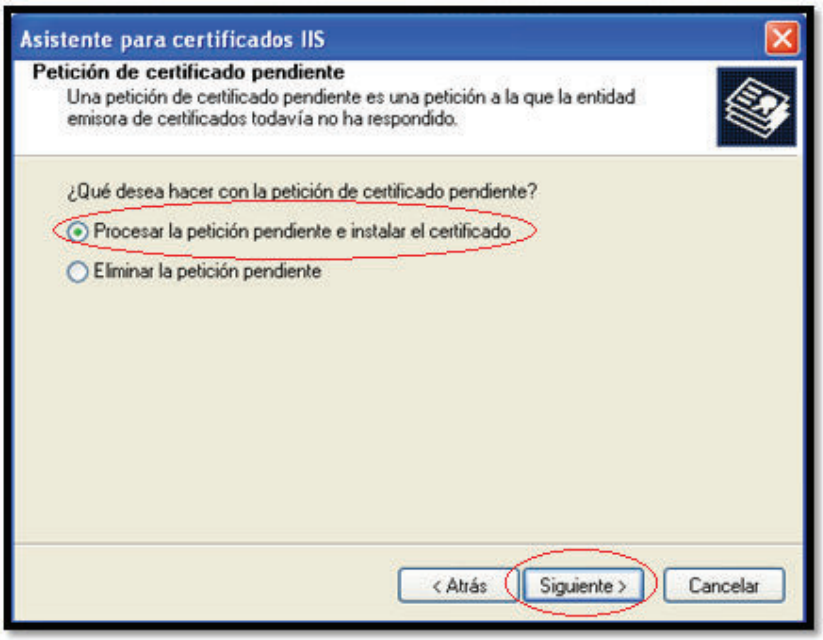

Figura 2.15 Petición e instalación de certificado

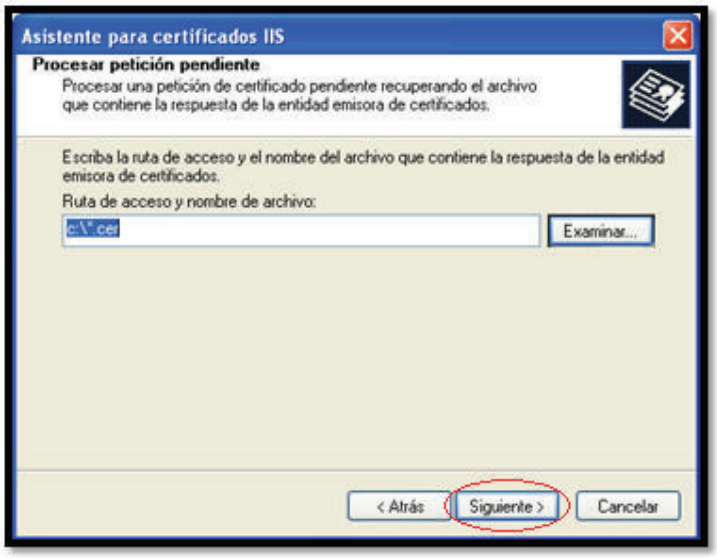

Figura 2.16 Ruta de acceso y nombre del archivo que contiene el certificado generado

2.3.2.3 Configuración de la dirección MAC

Una vez configurado los equipos virtuales, se procede a configurar los equipos físicos para que tengan las mismas direcciones MAC, para realizar este cambio en Windows se utilizó el programa EtherChange [30], para realizar el cambio se procede con los pasos que se muestran en la Figura 2.17. El proceso consiste en elegir la interfaz de red y especificar la nueva dirección MAC.

```
EtherChange 1.1 - (c) 2003-2005, Arne Vidstrom
                - http://ntsecurity.nu/toolbox/etherchange/

0. Exit
1. Atheros AR8151 PCI-E Gigabit Ethernet Controller (NDIS 6.20)
2. Realtek RTL8192CU Wireless LAN 802.11n USB 2.0 Network Adapter
3. Intel(R) Centrino(R) Wireless-N 2230
4. Apple Mobile Device Ethernet

Pick a network adapter: 1

0. Exit
1. Specify a new ethernet address
2. Go back to the built-in ethernet address of the network adapter

Pick an action: 1

Specify a new ethernet address (in hex without separators): 00000000000001
```

Figura 2.17 Programa EtherChange

2.3.3 CONFIGURACIÓN DEL SWITCH SDN

Para la configuración del switch primero se debe especificar la VLAN con los puertos que van a funcionar con el protocolo OpenFlow (VLAN10), para lo cual primero se crea la VLAN y se asocia a los puertos del switch, y una VLAN diferente por donde se conectará el controlador (VLAN1), al ser un switch de la casa fabricante HP se procedió como se muestra en el Segmento de código 2.4

```
openflow(conf)# vlan 10
openflow(vlan-10)# untagged ethernet 1,2,3,4
```

Segmento de código 2.4 Comandos para crear la VLAN en el switch HP

El controlador que se conecta al puerto 5 se mantendrá en la VLAN por defecto (VLAN 1). La Figura 2.18 presenta un listado de puertos y VLAN asignados a estas.

Una vez que se tiene la asignación de puertos se procede con la configuración del protocolo OpenFlow en el switch.

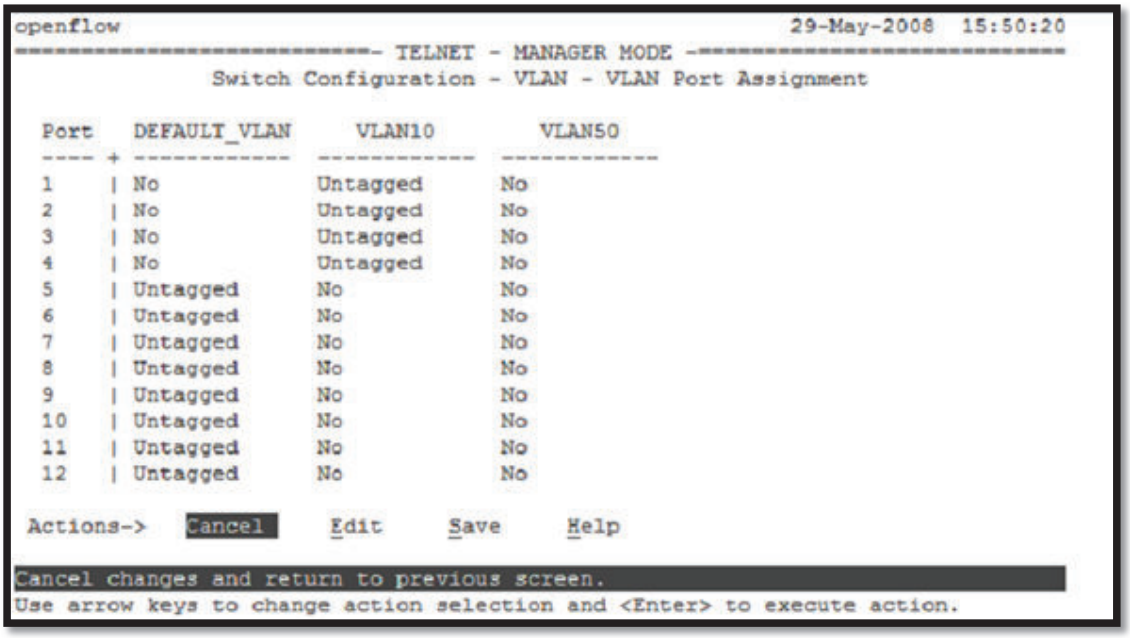

Figura 2.18 Puertos y VLANs asignados

Primero se debe configurar el controlador que instanciará las reglas, para lo cual se debe especificar la dirección IP del controlador, la VLAN en la que está el puerto por el que se conecta y un identificador, como se muestra en el Segmento de código 2.5.

```
openflow(conf)#openflow
openflow(openflow)#controller-id 1 ip 192.168.0.100 controller-
interface vlan 1
```

Segmento de código 2.5 Creación del identificador de controlador OpenFlow

Segundo se debe configurar la instancia con los parámetros adecuados para establecer la comunicación entre el switch y el controlador, como son; el puerto escucha por donde recibirá las reglas, la VLAN con los puertos OpenFlow, y asociar al controlador, con el identificador anteriormente creado, como se muestra en el Segmento de código 2.6

```
openflow(openflow)#instance prototipo
openflow(of-inst-prototipo)#listen-port 6633
openflow(of-inst-prototipo)#member vlan 10
openflow(of-inst-prototipo)#controller-id 1
```

Segmento de código 2.6 Configuración de los parámetros para OpenFlow

Una vez configurados los parámetros se procede a habilitar la instancia creada y el protocolo OpenFlow en el switch, de esta forma el switch queda a la espera de una conexión del controlador. Lo indicado se muestra en el Segmento de código 2.7.

```
openflow(of-inst-prototipo)#enable
openflow(of-inst-prototipo)#exit
openflow(openflow)#enable
```

Segmento de código 2.7 Comandos para habilitar la instancia y el soporte de OpenFlow

Se puede verificar las configuraciones con los comandos mostrados en el Segmento de código 2.8.

```
openflow(conf)#show openflow
openflow(conf)#show openflow instance prototipo
```

Segmento de código 2.8 Comandos para mostrar las configuraciones

El resultado debe ser similar al presentado Figura 2.19 y en la Figura 2.20.

```
openflow(openflow)# show openflow

OpenFlow                    : Enabled
IP Control Table Mode       : Disabled
Egress Only Ports Mode      : Disabled

Instance Information

                                                 No. of      No. of      OpenFlow
 Instance Name                    Oper. Status   H/W Flows   S/W Flows   Version
 -------------------------------- ------------ ----------- ----------- --------
 marlonof                         Up           0           0           1.0
 prototipo                        Up           1           0           1.0
```

Figura 2.19 Resultado del comando `show OpenFlow`

```
openflow(openflow)# show openflow instance prototipo

Configured OF Version    : 1.0
Negotiated OF Version    : 1.0
Instance Name            : prototipo
Admin. Status            : Enabled
Member List              : VLAN 10
Listen Port              : 6633
Oper. Status             : Up
Oper. Status Reason      : NA
Datapath ID              : 000a28924a7c5bc0
Mode                     : Active
Flow Location            : Hardware and Software
No. of Hw Flows          : 1
No. of Sw Flows          : 0
Hw. Rate Limit           : 0 kbps
Sw. Rate Limit           : 100 pps
Conn. Interrupt Mode     : Fail-Secure
Maximum Backoff Interval : 60 seconds
Probe Interval           : 10 seconds
Hw. Table Miss Count     : 101898
No. of Sw Flow Tables    : NA
Egress Only Ports        : None
Table Model              : Single Table

Controller Id Connection Status Connection State Secure Role
------------- ----------------- ---------------- ------ ------
1             Disconnected      Backoff          No     Equal
```

Figura 2.20 Resultado del comando `show openflow instance`

2.4 GENERACIÓN DE ATAQUES

La generación de los ataques se la realiza desde el equipo 4, que es el atacante, se utilizan algunas herramientas incorporadas en la distribución Kali Linux que facilitan en gran medida su ejecución.

2.4.1 GENERACIÓN DE ARP SPOOFING

Este ataque se lo hizo desde la distribución Kali Linux, con el comando `arpspoof` como se muestra en la Figura 2.21. Los parámetros de `arpspoof` son los que se muestran en Segmento de código 2.9 [31]:

```
arpspoof –i ethX –t IP1 IP2
```

Segmento de código 2.9 Comando para genera el ataque ARP SPOOFING

- **-i eth1**: La interfaz por la que está conectada a la LAN.
- **-t IP1**: Permite especificar la dirección IP a la que va dirigido el ataque.
- **IP2**: Permite especificar la dirección IP del equipo que se quiere suplantar.

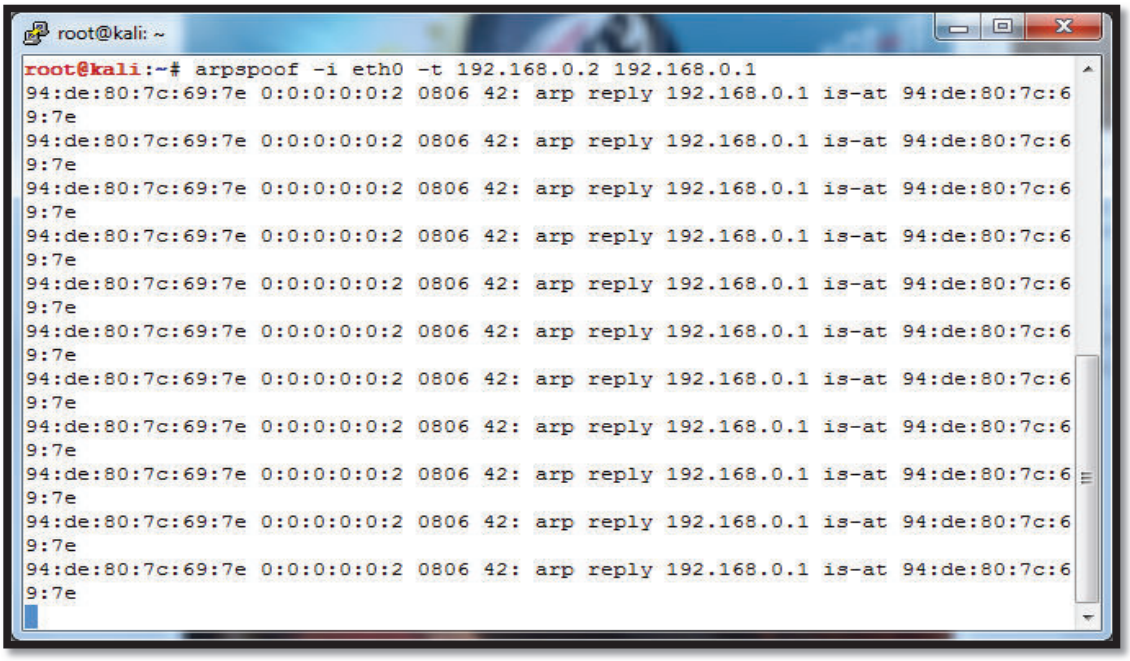

Figura 2.21 Generación del ataque ARP SPOOFING

2.4.2 GENERACIÓN DE IP SPOOFING

Este ataque se le puede generar de varias formas con la herramienta `hping3`, a continuación se muestran dos de ellas, una haciendo uso del protocolo ICMP y otra con el protocolo TCP como se muestra en las Figuras 2.22 y Figura 2.23. Los parámetros de `hping3` son los que se muestran en el Segmento de código 2.10:

```
Hping3 <-1 ó -S> IP1 -a IP2 -c <Número>
```

Segmento de código 2.10 Comando para genera el ataque IP SPOOFING

- **-1**: Especifica el protocolo ICMP
- **-S**: Especifica el protocolo TCP
- **IP1**: Indica la dirección IP de la víctima
- **-c <Número>:** Número de paquetes que se van a generar
- **-a IP2**: Indica la dirección IP suplantada

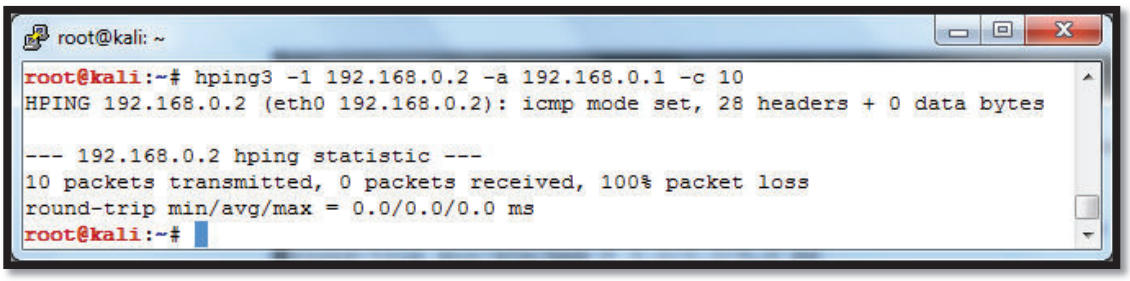

Figura 2.22 Generación de IP SPOOFING usando el protocolo ICMP

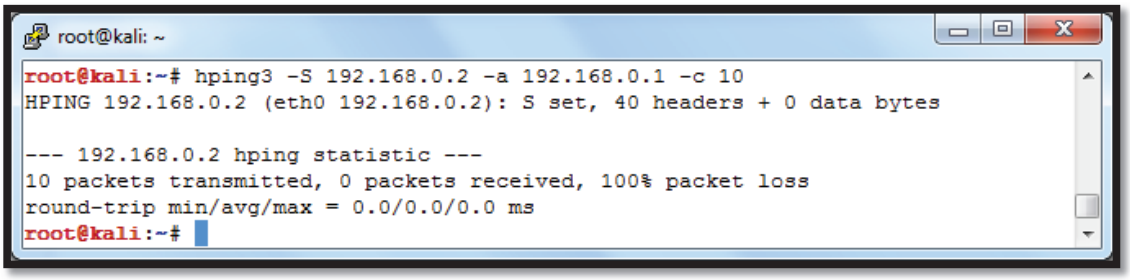

Figura 2.23 Generación de IP SPOOFING usando el protocolo TCP

2.4.3 GENERACIÓN DE DNS SPOOFING

Para este ataque se utilizó la herramienta `ettercap` que está incluida en la distribución Kali Linux como se muestra en la Figura 2.24. Los parámetros se muestran en el Segmento de código 2.11 [32].

```
ettercap -Tqi ethX -P dns_spoof -M arp // //
```

Segmento de código 2.11 Comando para genera el ataque DNS SPOOFING

Cada uno de los parámetros que se utiliza en el comando, cumplen la función que se explica a continuación:

- **-T**: Especifica el uso de la interfaz basada en texto
- **-q**: Ejecuta comandos en modo silencioso

- **-i eth1**: Elección de la interfaz por la que se conecta a la LAN
- **dns_spoof -P**: Especifica el uso del plug-in dns_spoof
- **-M arp**: Inicia un ataque *MAN-IN-THE-MIDDLE* de envenenamiento ARP para interceptar los paquetes entre *hosts*
- *// //*: Especifica como objetivo de ataque a toda la red

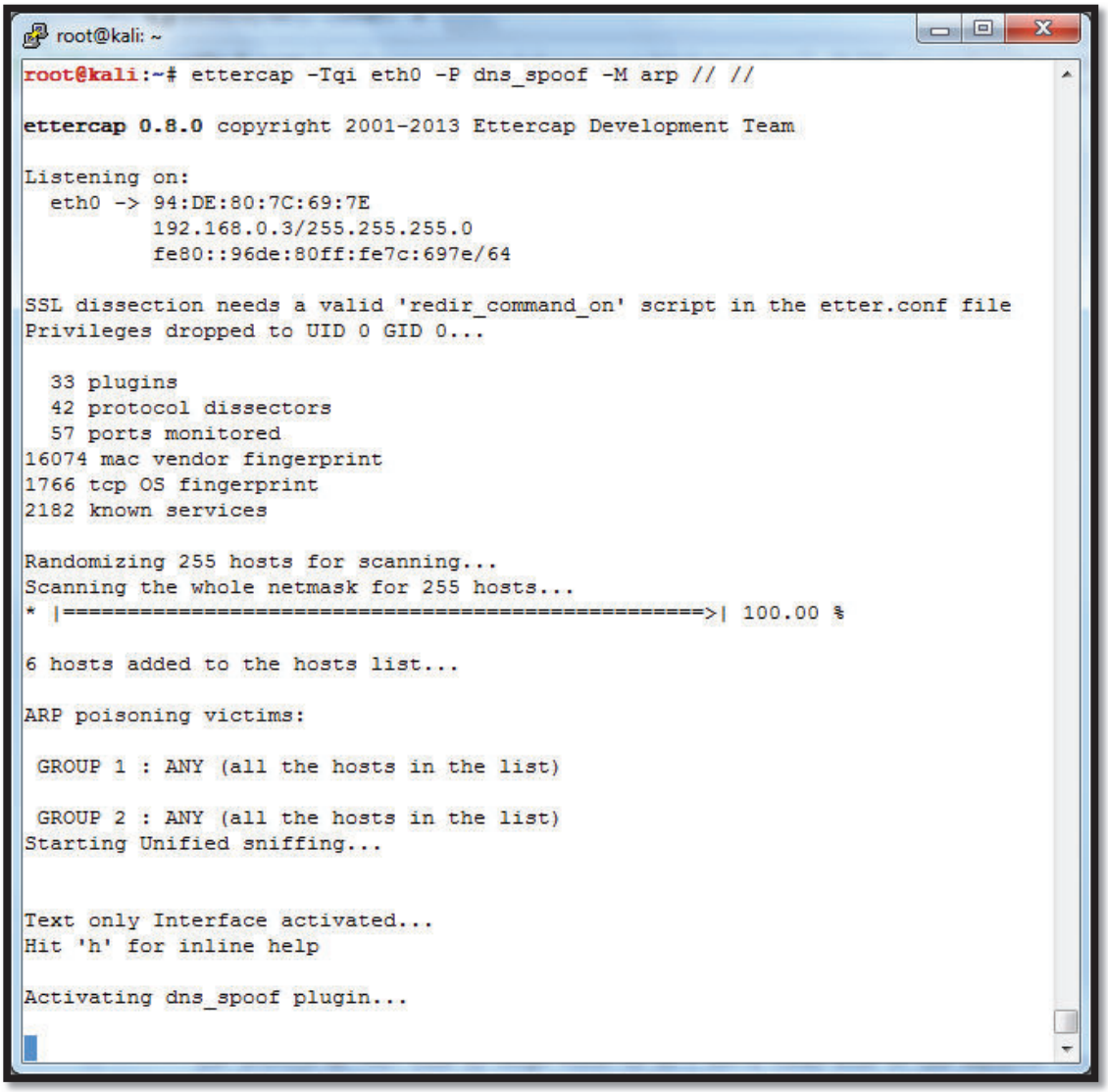

Figura 2.24 Generación del ataque DNS SPOOFING

2.4.4 GENERACIÓN DE TCP SYN FLOOD

Para generar este ataque se utilizó la herramienta `hping3`, como se muestra en la Figura 2.25, en la que se especifica los parámetros como se muestra en el Segmento de código 2.12 [33]:

```
hping3 -p <Puerto> -S --flood IP
```

Segmento de código 2.12 Comando para genera el ataque TCP SYN FLOOD

- **-S**: Define la bandera SYN que se va a activar en el protocolo TCP.
- **-i u1**: Específica cada qué tiempo se enviara un paquete en este caso cada 1 micro segundo.
- **-p <Puerto>**: Elige el puerto al que se desea llegar, en este caso 80.
- **IP**: Especifica dirección IP de la víctima.

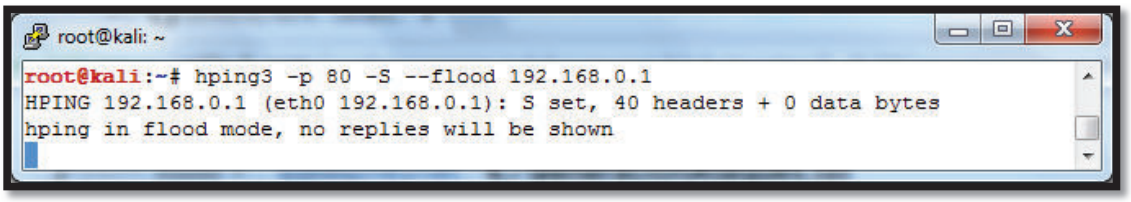

Figura 2.25 Generación del ataque TCP SYN FLOOD

2.4.5 GENERACIÓN DE THC SSL DOS

Para generar este ataque se utiliza la herramienta `thc-ssl-dos`, como se muestra en la Figura 2.26, los parámetros se muestran en el Segmento de código 2.13 [34] :

```
thc-ssl-dos --accept IP <Puerto (opcional)>
```

Segmento de código 2.13 Comando para genera el ataque THC SSL DoS

- **IP**: Dirección IP de la víctima, recordar que corresponde a la de un servidor con soporte HTTPS.
- **--accept**: Permite que el usuario que realiza la prueba de estrés acepte el término de uso del *script* en cuestión.
- **<Puerto>**: Puerto en que escucha HTTPS (443), este campo es opcional.

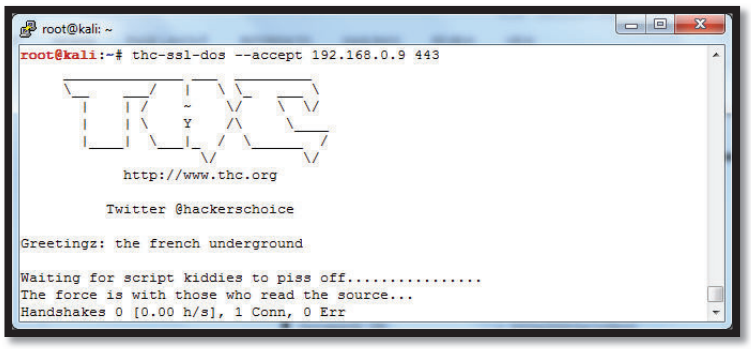

Figura 2.26 Generación del ataque THC SSL DOS

2.4.6 GENERACIÓN DE SMURF

Con la herramienta `hping3` se puede generar este ataque, como se muestra en la Figura 2.27, en el que se especifican los parámetros en el Segmento de código 2.14:

```
hping3 -1 --flood IP1 -a IP2
```

Segmento de código 2.14 Comando para genera el ataque `SMURF`

- **-1**: Para indicar que se trata de un paquete ICMP
- **--flood**: Le indica a hping que envié los paquetes a la máxima velocidad posible.
- **IP1**: La dirección IP de la víctima
- **–a IP2**: La dirección IP de origen que en este caso es la dirección de *broadcast*.

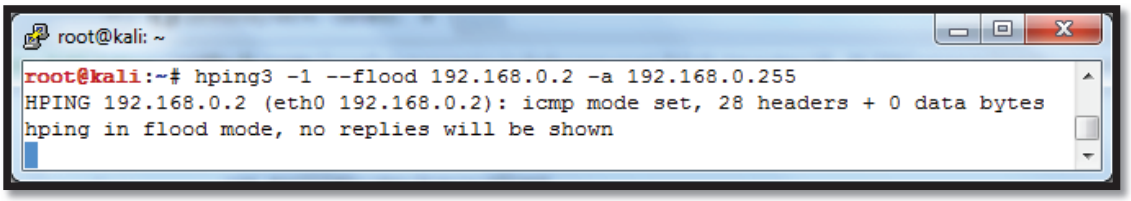

Figura 2.27 Generación del ataque `SMURF`

CAPÍTULO III

3 DISEÑO DE LA APLICACIÓN

En este capítulo se presentada el diseño de la aplicación desarrollada para el controlador Pyretic, el mismo que abarca el diagrama de bloques, los diagramas de secuencias, los diagramas de flujo, entre otros.

Adicionalmente se realiza un estudio más a fondo del controlador utilizado, en el que se especifica los módulos que lo conforman, las formas de funcionamiento y algunos ejemplos de operaciones básicas que son propias del controlador.

3.1 CONTROLADOR PYRETIC

En la primeras generaciones de SDN las plataformas de los controladores ofrecían diversas API[48] de programación a muy bajo nivel, realmente muy parecidas a las interfaces de los conmutadores, lo que conllevaba a que los programadores tengan que desarrollar sus aplicaciones en lenguaje *assembler*[49], manipulando patrones de bits de paquetes y manejando cuidadosamente las tablas de reglas que se utilizaban.

En el "Proyecto Frenetic"[50] se ha diseñado un sistema de tiempo de ejecución simple, reusable y de un alto nivel de abstracción para programar las SDN, que genera e instala automáticamente las reglas de bajo nivel correspondientes en cada switch. Dicha abstracción cubre las principales facetas que conlleva la administración de las especificaciones de una red como son: las políticas de envío de paquetes, el monitoreo de las condiciones de la red, y las políticas de actualización automática para responder a diferentes eventos generados en la red [35].

El resultado del mencionado proyecto es el controlador Pyretic, una plataforma de programación pasada en el lenguaje Python que eleva el nivel de abstracción y

---

[48] API (*Application Programming Interface*) es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
[49] *Assembler:*, es un lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables
[50] Proyecto Frenetic:, es un proyecto colaborativo entre las universidades de Cornell y Princeton para desarrollar lenguajes de programación para SDNs.

habilita la creación de software modular, permitiendo a los programadores crear sofisticadas aplicaciones para Redes Definidas por Software [36].

Pyretic es de código abierto y tiene una licencia del estilo BSD[51].

3.1.1 ESTRUCTURA DE MODULOS

El *core* o núcleo del controlador Pyretic, está constituido por varios módulos que en conjunto le permiten cumplir con ciertas políticas propias y más fáciles de utilizar al momento de desarrollar las distintas aplicaciones, entre los cuales se tiene [37]:

- Módulo `CLASSIFIER`
- Módulo `LANGUAGE`
- Módulo `NETWORK`
- Módulo `PACKET`
- Módulo `RUNTIME`

3.1.1.1 Módulo CLASIFIER

Este módulo contiene una lista de reglas, donde el orden de la lista implica las prioridades relativas de las reglas. Semánticamente, los clasificadores son funciones de paquetes para conjuntos de paquetes, similar al flujo de las tablas OpenFlow [38].

3.1.1.2 Módulo LANGUAGE

El módulo `LANGUAGE` es el módulo en el que se definen las sintaxis propias del lenguaje, es decir se implementan variables estándar por ejemplo para ciertos campos de los paquetes que se reciban en el controlador. A continuación se muestra algunos de los campos [39]:

- De la cabecera:
  - ✓ `srcmac`.- Dirección MAC del equipo donde se origina el paquete.
  - ✓ `dstmac`.- Dirección MAC del equipo al que está dirigido el paquete.
  - ✓ `srcip`.- Dirección IP del equipo que origina el paquete.
  - ✓ `dstip`.- Dirección IP del equipo al que está dirigido el paquete.

[51] BSD (*Berkeley Software Distribution*) es una licencia de software libre permisiva similar a la licencia de OpenSSL.

- ✓ `tos`.- Para el caso de los protocolos que tengan el campo tipo de servicio.
- ✓ `srcport`.- Para los protocolos TCP y UDP, corresponde al puerto origen del paquete.
- ✓ `dstport`.- Para los protocolos TCP y UDP, corresponde al puerto destino del paquete.
- ✓ `ethtype`.- Se especifica que protocolo de capa 3 se manejará, ejemplo 0x800 corresponde al protocolo IP.
- ✓ `protocol`.- Depende del protocolo, específica a que hace referencia cada paquete, por ejemplo *`echo request`*.

- De etiquetado:
  - ✓ `vlan_id`.- Es un identificador de la VLAN en la que se esté trabajnado.
- De localización:
  - ✓ `Switch`.- Es el campo que permite identificar a los switches de la red.
  - ✓ `Inport`.- Puerto del switch por el que llego el paquete.
  - ✓ `Outport`.- Puerto del switch por el que se enviará el paquete.
- De contenido:
  - ✓ `Raw`.- Corresponde al paquete original (sin las cabeceras del paquete OpenFlow),
  - ✓ `header_len`.- Especifica la longitud de la cabecera OpenFlow del paquete,
  - ✓ `payload_len`.- Especifica la longitud del paquete sin cabecera OpenFlow.

También se definen algunos métodos útiles al momento de implementar nuevas aplicaciones como son:

- `flood()`.- Método que envía el paquete por todos los puertos OpenFlow del switch, menos por el que ingresó el paquete al switch.
- `filter()`.- Método que bloquea o filtra los paquetes especificados.
- `drop()`.- Método que elimina o descarta los paquetes que cumplan una cierta condición.

3.1.1.3 Módulo NETWORK

Permite establecer o realizar un esquema general de la estructura lógica de la red, es decir, que topología se tiene y cómo interactúan cada uno de los elementos que la constituyen, en este módulo están definidos los distintos tipos de direcciones, físicas y lógicas [40].

3.1.1.4 Módulo PACKET

Este módulo está encargado de dar formato a los paquetes que se reciban y se envían desde y hacia el controlador, así pues, este módulo deberá verificar que los paquetes recibidos se encuentren íntegros, que los paquetes enviados sean estructurados adecuadamente, esto debe hacer dependiendo del tipo de paquete que se reciba de la red de datos. Este módulo es el encargado de encapsular y desencapsular los paquetes OpenFlow [41].

3.1.1.5 Módulo RUNTIME

Es un módulo definido directamente en el código fuente del controlador Pyretic y se encarga de las principales tareas que se pueden llevar a cabo en tiempo de ejecución [42].

3.1.2 FUNCIONAMIENTO DEL CONTROLADOR PYRETIC

El controlador Pyretic para poder iniciarse de forma correcta, necesita que se especifique ciertos requisitos que le permiten realizar las tareas requeridas. Al iniciarse busca todos los módulos necesarios para ello, y posteriormente arranca.

3.1.2.1 Ejecución del Controlador

Para ejecutar el controlador Pyretic, se requiere especificar el archivo que inicia el controlador, seguido del modo de operación y finalmente el modulo que la aplicación ejecutará en el controlador.

Desde la línea de comandos se debe ubicar en el directorio en donde se encuentra el archivo `pyretic.py`, que es el archivo que inicia la ejecución del controlador Pyretic, dicho archivo se encuentra en el directorio en el que se ha instalado el controlador, que por lo general se trata del directorio: `/home/mininet/pyretic`. Luego de lo cual debe especificarse el modo de operación del controlador.

Pyretic puede funcionar en 3 modos diferentes:

- Interpretado (`i`): En este modo el controlador instala una sola regla en el switch, haciendo que cada paquete que llegue al switch sea enviado al controlador para ser analizado, es el modo más lento por el mismo hecho de que cada paquete se analiza (depura). Este modo es el que se lo utilizará en este proyecto ya que al tratarse de seguridades no se puede estar al tanto del ataque con el primer paquete que llegue, si no que tendrán que pasar los paquetes necesarios antes de darse cuenta de que se trata de un ataque, y al fin bloquearlo o en este caso conocer el origen del paquete para que cada paquete sea enviado a la *honeynet*.
- Reactivo (`r0`).- En este modo el controlador instala las reglas de forma reactiva[52] en el switch, basado en las políticas y los paquetes enviados.
- Proactivo (`p0`).- En este modo el controlador instala las reglas de forma proactiva[53] en el switch, basado en las políticas y los paquetes enviados [43].

Finalmente se debe especificar la dirección completa del archivo en donde está la aplicación escrita en Python que ejecutará el controlador, por ejemplo el archivo: `/pyretic/sdnhonynet/controlador.py`.

Es importante notar que al ejecutar la aplicación se debe especificar la dirección separada por ”.“ en vez de ”/”, y en el archivo de Python debe omitirse la extensión del archivo (`.py`) para que no se produzca ningún error. En el Segmento de código 3.1 se indica al controlador Pyretic que trabaje en el modo interpretado y que ejecute a la aplicación `controlador.py`:

```
$ pyretic.py -m i pyretic.sdnhoneynet.controlador
```

Segmento de código 3.1 Ejemplo de ejecución del controlador Pyretic

[52] Reactiva, se trata a los paquetes dependiendo de las reglas instaladas en el switch.
[53] Proactiva, aprende de los paquetes que le llega (direcciones destino) para posteriormente tratar a los paquetes de una forma más rápida.

3.1.2.2 Método Main

Cada programa de Pyretic debe tener un método `main` y como mínimo debe importar la librería principal del núcleo de Pyretic denominada `pyretic.lib.corelib`. El método `main` debe devolver una instancia de la clase `policy` de Pyretic (o cualquier clase derivada). Por ejemplo, uno de los programas Pyretic más simples que se puede escribir se presenta en el Segmento de código *3.2* [44].

```
from pyretic.lib.corelib import *
def main ():
   return flood()
```

Segmento de código 3.2 Ejemplo de una aplicación simple para el controlador Pyretic

En general es probable que se desee importar la biblioteca estándar Pyretic ya que contiene algunas funciones útiles para establecer puntos de interrupción, las funciones de impresión útiles, y algunas políticas de filtro de conveniencia. En el Segmento de código *3.3* se presenta la línea de código que permite importar dicha biblioteca.

```
from pyretic.lib.std import *
```

Segmento de código 3.3 Importación de la biblioteca estándar de Pyretic

3.1.2.3 Definiciones propias de PYRETIC

Como se ha mencionado anteriormente, Pyretic define en el módulo `LANGUAGE` ciertas políticas de implementación que facilitan a los programadores el desarrollo de diferentes aplicaciones, entre las principales se tiene las indicadas en la Tabla 3.1 [45].

POLíTICA	SINTAXIS	DESCRIPCIÓN	EJEMPLO

match	`match(f=v)`	Devuelve un conjunto que contiene un paquete si el valor del campo `f` coincide con el valor `v`, caso contrario devuelve un conjunto vacío.	`match(dstmac=EthAddr('00:00:00:00:00:01'))`
drop	`drop`	Devuelve un conjunto vacío	`drop`
identity	`identity`	Devuelve un conjunto que es copia del paquete	`identity`
modify	`modify(f=v)`	Devuelve una copia del paquete donde el valor `f` ha sigo reasignado con el valor `v`	`modify(srcmac=EthAddr('00:00:00:00:00:01'))`
forward	`fwd(a)`	Devuelve una copia del paquete donde el campo puerto de salida es configurado al valor `a`	`fwd(a)`
flood	`flood()`	Devuelve una copia del paquete que es enviada por cada puerto del *spanning tree.*	`flood()`
parallel composition	`A + B`	Devuelve la unión de la salida de `A` con la salida de `B`	`fwd(1) + fwd(2)`
sequential composition	`A >> B`	Devuelve la salida de `B` donde la salida de `A` es la entrada de `B`	`modify(dstip=IPAddr('10.0.0.2')) >> fwd(2)` `match(switch=1) >> flood()`
negation	`~A`	Devuelve una negación lógica	`~match(switch=1)`

Tabla 3.1 Definiciones propias del controlador Pyretic

3.2 IMPLEMENTACIÓN DE LA APLICACIÓN

Actualmente existe una gran variedad de herramientas informáticas que permiten generar diferentes tipos de ataques, inclusive existen distribuciones de Linux que han sido optimizadas para este tipo de procesos, como es el caso de BackTrack que posteriormente fue rediseñada y renombrada como Kali Linux, que es

precisamente la distribución que ha sido utilizada en este Proyecto para generar los diferentes ataques [46], [47].

Una vez que se han explicado las topologías que se utilizarán para la implementación de este prototipo, así como también la implementación y configuración de los diferentes equipos de red que se necesitará, se presentará el diseño e implementación de la aplicación que correrá el controlador y que permitirá realizar la detección de los ataques ya especificados anteriormente.

3.2.1 DIAGRAMA DE BLOQUES DE LA APLICACIÓN

La aplicación que ejecutará el controlador, consta de varios módulos, entre ellos figura un módulo principal que es el que correrá específicamente el controlador y este se encarga de enviar cada paquete recibido hacia los diferentes módulos de procesamiento, de ser el caso este módulo principal adicionalmente recibirá la respuesta de cada módulo de procesamiento y procederá a llamar al módulo `enviar.py`, ya sea para dejarlo pasar a la LAN o para desviarlo a la *honeynet*.

El diagrama de bloques del módulo principal se presenta en la Figura 3.1. El módulo principal se denomina `controlador.py`, y en este se cargan los parámetros (librerías, diccionarios, listas, etc.), dichos parámetros solo se cargan la primera vez que se ejecuta el controlador, para luego proceder a procesar el paquete y extraer los diferentes campos que contiene.

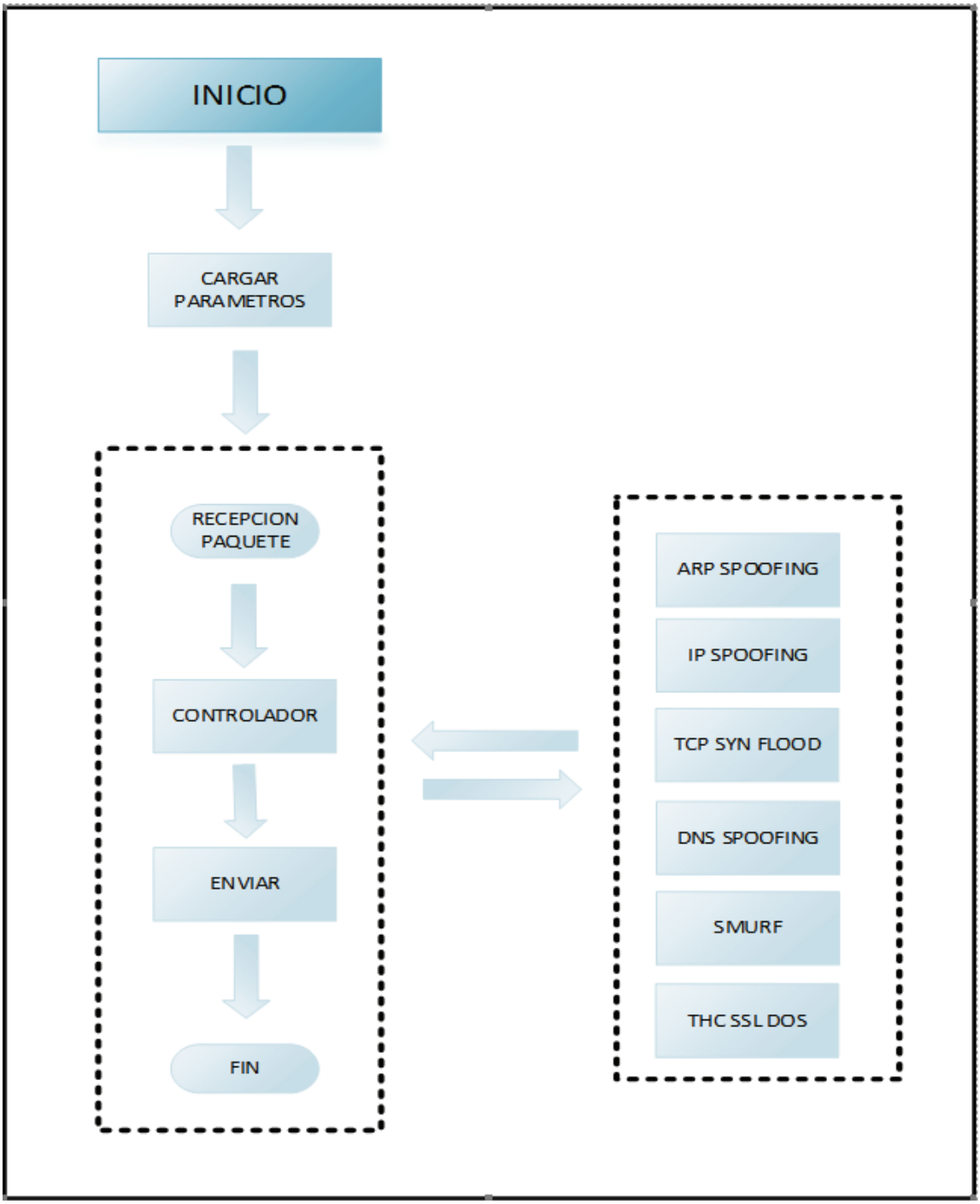

Figura 3.1 Diagrama de bloques del módulo principal

A continuación, se observa en el diagrama el módulo denominado `controlador`, es el módulo central de la aplicación, este es el que recibe y procesa cada uno de los paquetes que le llegan al switch, debe recibir como entrada la opción que especifica que módulo es el que se procesará, cuando la opción es igual a 0, la

aplicación ejecuta secuencialmente los bloques necesarios dependiendo del tipo de paquete que se haya recibido, si la opción es diferente de 0, el controlador únicamente procesará el bloque correspondiente a la opción especificada. Las opciones de ejecución son las siguientes:

Opción = 1, ejecuta el método `arp_spoofing` del módulo `arp.py (arp.arp_spoofing)`.

Opción = 2, ejecuta el método `ip_spoofing` del módulo `ip.py (ip.ip_spoofing)`.

Opción = 3, ejecuta el método `tcp_syn_flood` del módulo `tcp.py (ip.ip_spoofing)`.

Opción = 4, ejecuta el método `dns_spoofing` del módulo `udp.py (udp.dns_spoofing)`.

Opción = 5, ejecuta el método `smurf` del módulo `icmp.py (icmp.smurf)`.

Opción = 6, ejecuta el método `thc_ssl_dos` del módulo `https.py (https.thc_ssl_dos)`.

Una vez se haya procesado el o los bloques necesarios, se devuelve al módulo del controlador una variable del tipo `string` que indica a donde se debe enviar el paquete analizado; la variable también puede estar vacía, lo que indica que al paquete se lo dejará de procesar.

3.2.2 DIAGRAMAS DE FLUJO

Los diagramas de flujo son representaciones esquemáticas gráficas, que simbolizan las ideas y conceptos relacionados, y que frecuentemente son usados para representar algoritmos.

A continuación se presentan los diagramas de flujo de los diferentes módulos que utilizará la aplicación desarrollada para el controlador Pyretic en este prototipo, así como una breve explicación de los mismos.

3.2.2.1 Diagrama de flujo del módulo controlador.py:

El diagrama de flujo del módulo `controlador.py` se presenta en la Figura 3.2 y la Figura 3.3, en ellas se representa el tratamiento que se le da a cada paquete de datos que se reciba en el controlador.

Al iniciar el controlador Pyretic, se especificará que debe iniciarse el modulo principal de la aplicación, este módulo empieza inicializando las variables que se requerirán en los demás módulos de procesamiento, adicionalmente se cargan algunos parámetros de configuración y valores de variables fijas desde un archivo de configuración y enseguida el modulo queda listo para que se reciba y procese cada paquete que reciba.

En la Figura 3.2 se muestra el proceso que sigue cada uno de los paquetes al llegar al módulo principal.

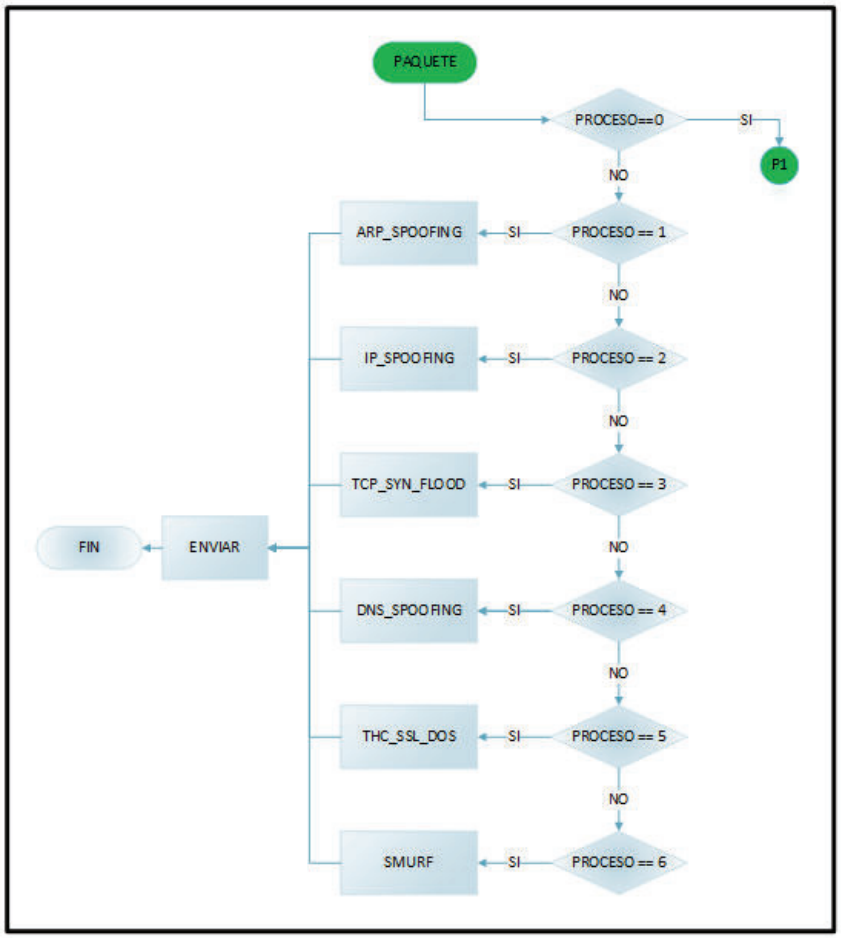

Figura 3.2 Diagrama del módulo `controlador.py` (Sección Primera)

Dependiendo del módulo de procesamiento que se haya especificado en el archivo de configuración `honeynet.cfg` el paquete se enviará a uno o varios módulos de procesamiento.

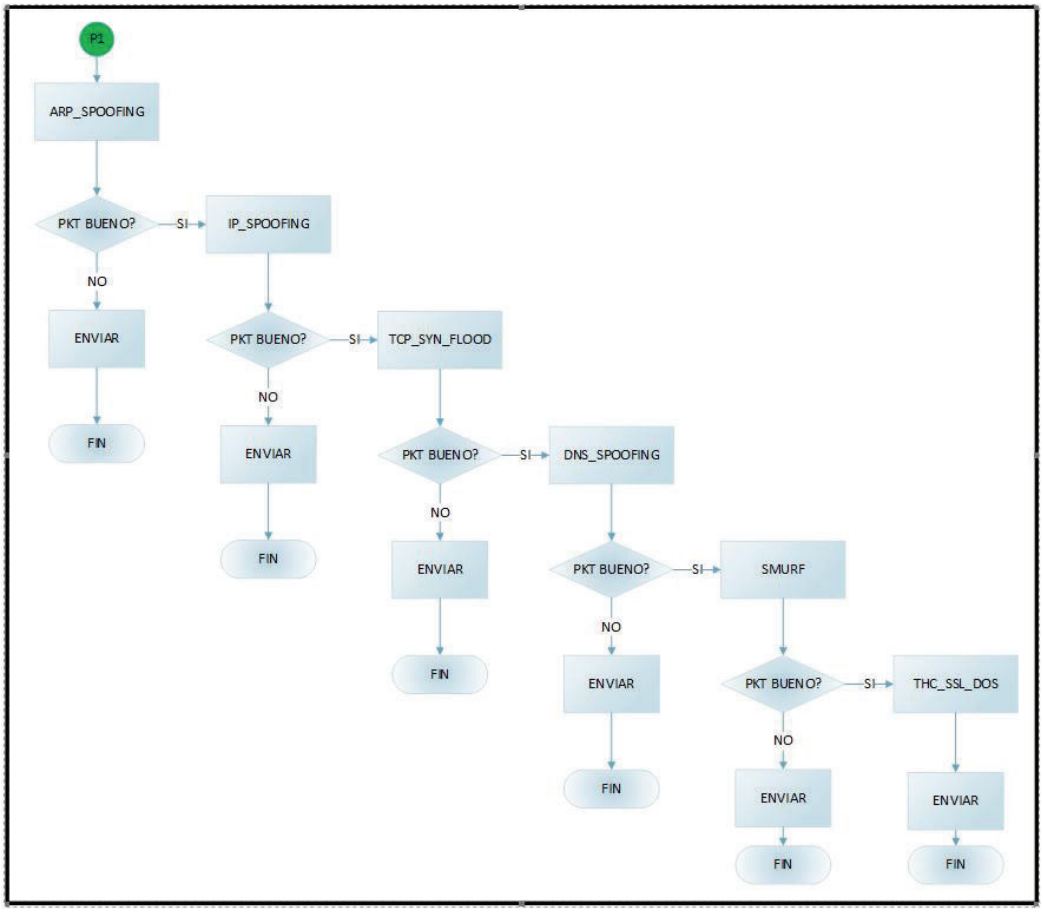

Figura 3.3 Diagrama del módulo `controlador.py` (Sección Segunda)

Si se configura la variable `PROCESO` con un valor entre 1 y 6, cuando se reciba el paquete se envía a un solo módulo de procesamiento (es decir se puede detectar un solo ataque informático), el correspondiente de acuerdo a lo especificado en la sección 3.2.1. Sin embargo, si a la variable `PROCESO` se le asigna el valor 0, entonces se procederá a ejecutar secuencialmente los módulos que fueren necesarios dependiendo del tipo de paquete que se reciba como se muestra en la Figura 3.3, por ejemplo si es un paquete ARP, se lo procesará únicamente en el módulo `ARP_SPOOFING` pero si es un paquete TCP, entonces se lo procesará en los módulos `IP_SPOOFING` y `TCP_SYN_FLOOD` y aunque tiene que inicialmente

pasar por el módulo ARP_SPOOFING, ese paquete no se lo procesará ahí, pues en dicho modulo solo se procesan paquetes del tipo ARP.

De la misma manera, un paquete del tipo HTTPS deberá atravesar todos los demás módulos de procesamiento antes de llegar al módulo de THC_SSL_DoS, pero debido a que esta encapsulado en un segmento TCP y este a su vez en un paquete IP, tanto el módulo IP_SPOOFING como el módulo TCP_SYN_FLOOD lo procesarán, lo que conlleva a que los módulos ARP_SPOOFING, DNS_SPOOFING y SMURF lo dejarán pasar de forma transparente.

3.2.2.2 Diagrama de flujo del módulo arp.py

Se muestra en la Figura 3.4 el diagrama de flujo del módulo arp.py que permite detectar el ataque ARP_SPOOFING. En este módulo se utilizan los siguientes arreglos:

- dicMacIp.- Que en condiciones normales, contiene las direcciones IP y MAC del equipo que origino el paquete.
- dicSolicitudes.- Contiene la dirección IP del equipo al que está dirigido la solicitud, en conjunto con el número de solicitudes enviadas.
- dicMacPuerto.- Diccionario que contiene la dirección MAC de los equipos con el puerto por el que se conecta al switch.
- lstAtacantes.- Lista que contiene las direcciones MAC de los equipos que han sido determinados como atacantes.

Cuando se recibe un paquete se verifica en un inicio si el paquete corresponde al tipo ARP. Si no es el caso, se verifica si el paquete fue enviado hacia un atacante, entonces se lo procede a enviar únicamente por el puerto del switch SDN en donde se encuentre conectado dicho atacante. Pero si el paquete recibido proviene de un atacante se lo envía a la *honeynet*.

Por otro lado, si se determina que efectivamente el paquete que se recibió es del tipo ARP, entonces se discrimina entre los distintos tipos de paquetes ARP.

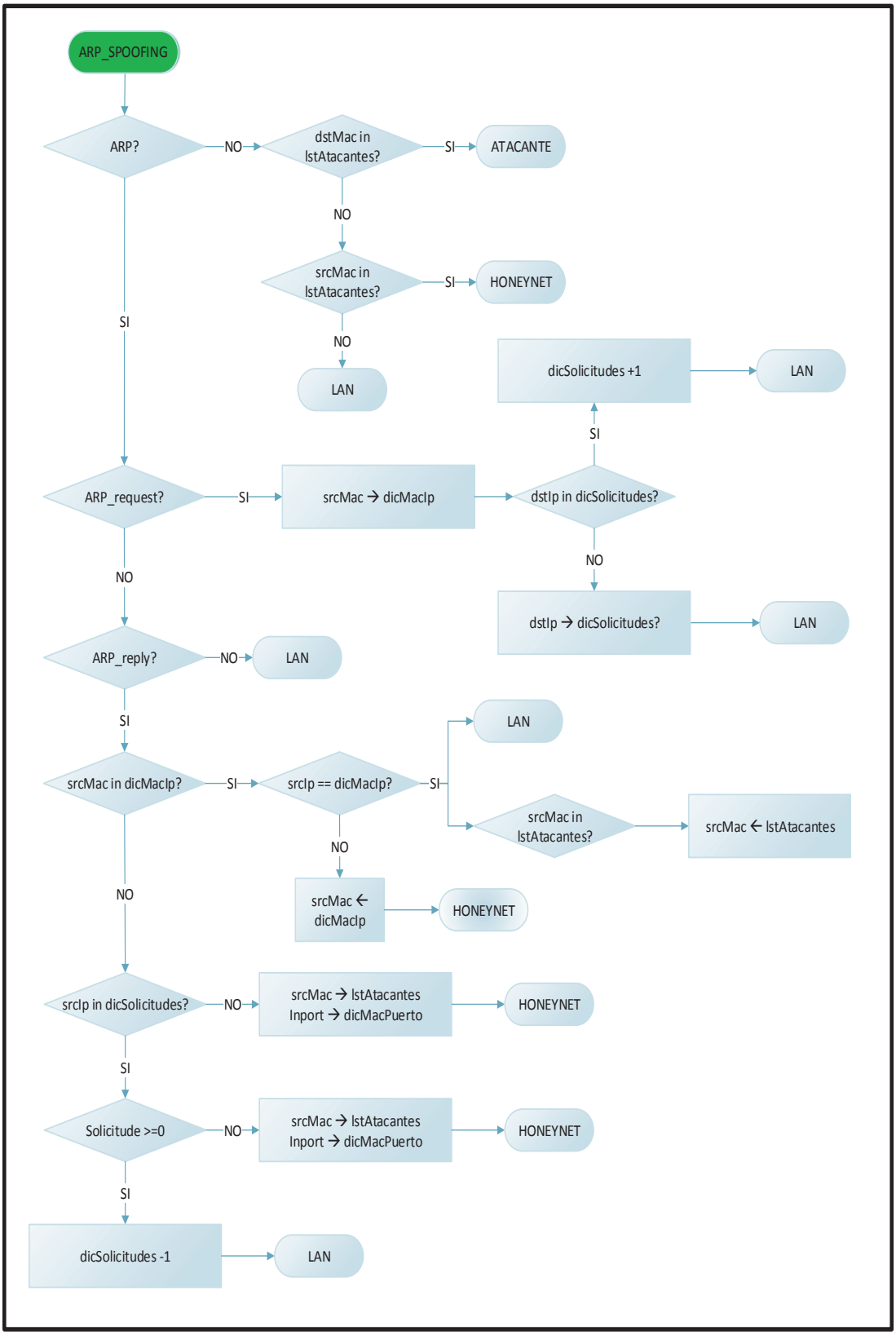

Figura 3.4 Diagrama de flujo del módulo `arp.py`

Si se trata de un paquete ARP_REQUEST[54], entonces se lo procede a agregar al diccionario dicMacIp, pues se lo toma como un paquete legítimo (no alterado por el atacante) inclusive si proviniera de la máquina del atacante, esto debido a que este ataque consiste únicamente en enviar paquetes del tipo ARP_REPLY[55], por lo tanto cualquier otro tipo de paquetes deberían tener sus verdaderas direcciones tanto IP como MAC, entonces el diccionario dicMacIp va a contener siempre información correcta y actual, pues cada vez que llegue un paquete de este tipo sobrescribirá el registro existente en caso de que ya exista o bien se lo agregará al diccionario si fue el primer paquete en llegar desde un origen en particular.

Si por otra parte llega un paquete del tipo ARP_REPLY, en primera instancia se verifica si la dirección MAC de origen ya se encuentra en el diccionario dicMacIp, de ser el caso se comprueba si la direcciona IP asociada a la dirección MAC del registro en el diccionario es igual a la direcciona IP de origen del paquete recibido. De ser así, se toma el paquete como un paquete legítimo, se lo envía a la LAN y adicionalmente se comprueba si la direcciona MAC de origen estaba en la lista de atacantes para proceder a eliminar el registro.

En el caso de que las direcciones IP del registro en el diccionario y del paquete recibido no sean iguales se procede a eliminar la entrada del diccionario, para dar una segunda oportunidad, puesto que podría deberse a que el servidor DHCP de la LAN le asigno una dirección IP nueva y por lo tanto se intenta no confundirse con un ataque, el paquete será enviado a la *honeynet* para evitar cualquier riesgo dentro de la LAN.

Si por el contrario, la dirección MAC del paquete ARP_REPLY recibido no tiene ningún registro en el diccionario dicMacIP (lo que significa que el computador que origino el paquete recibido, posiblemente el atacante, aún no ha realizado ningún ARP_REQUEST), se procede a verificar si existen solicitudes ARP_REQUEST a la dirección IP de la cual proviene el paquete recibido. Si no existen solicitudes, se lo clasifica inmediatamente como atacante (puesto que es imposible recibir respuestas si no existieron solicitudes) por lo que se agrega la dirección MAC de

[54] ARP_REQUEST Paquete que corresponde a una solicitud del protocolo ARP.
[55] ARP_REPLAY: Paquete que corresponde a una respuesta del protocolo ARP.

origen a la lista de atacantes y se guarda el puerto por el que llego el paquete al switch en el diccionario `dicMacPuerto`, eso ayuda a que cuando lleguen paquetes destinados al atacante se envía únicamente al mismo y no a toda la LAN o a la *honeynet*, y finalmente se envía el paquete a la *honeynet*.

En el caso de que si existieron solicitudes `ARP_REQUEST`, adicionalmente se debe verificar si el número de solicitudes es mayor a cero, puesto que por cada respuesta se disminuye en uno el número de solicitudes existentes, si se llegara a recibir más respuestas que el número de solicitudes, esto también indicará que se trata de un ataque por lo que se procede al igual que en al caso anterior a agregar tanto la direcciona MAC origen a la lista de atacantes como el número de puesto al diccionario `dicMacPuerto`.

3.2.2.3 Diagrama de flujo del módulo ip.py

Cuando se recibe un paquete en el módulo de procesamiento `ip.py`, se busca determinar si se trata de un ataque de `IP_SPOOFING`, por lo que se procede a realizar el análisis mostrado en la Figura 3.5.

En este módulo se utilizan los siguientes arreglos:

- `dicIpMac.-` Diccionario que contiene las direcciones IP y MAC del equipo que origino el paquete.
- `dicMacPuerto.-` Diccionario que contiene la dirección MAC de los equipos con el puerto por el que se conecta al switch.
- `lstMacAtacantes.-` Lista que contiene las direcciones MAC de los equipos que han sido detectados como atacantes.

Inmediatamente al llegar un paquete, se verifica si la dirección MAC origen se encuentra en la lista de atacantes, de ser así, se desvía el paquete hacia la honeynet y se evita un mayor procesamiento.

Si por otro lado, la dirección MAC origen del paquete recibido no está en la lista de atacantes, se debe determinar el tipo de paquete que es.

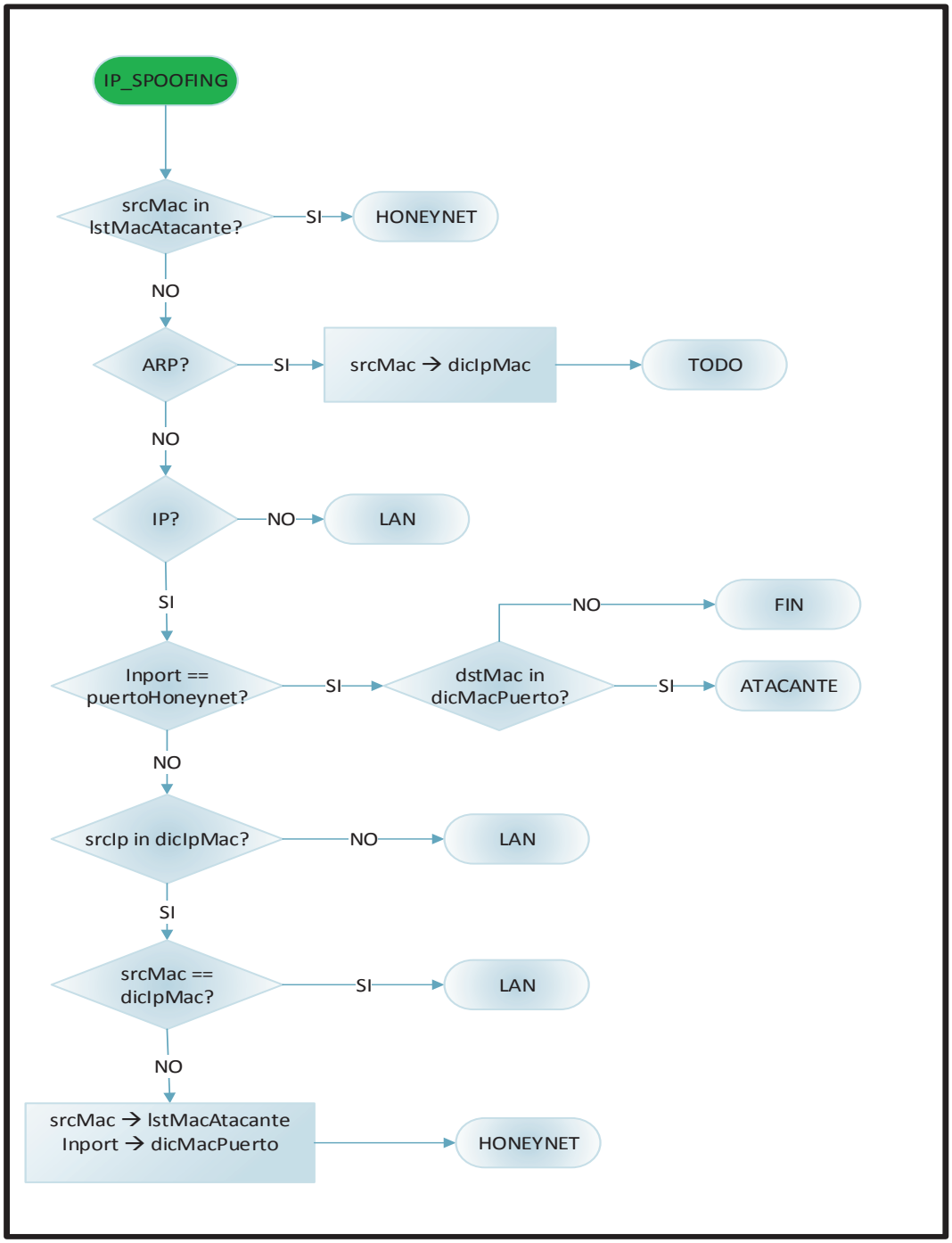

Figura 3.5 Diagrama de flujo del módulo `ip.py`

Si se trata de un paquete del tipo ARP, entonces se agrega un nuevo registro en el diccionario `dicIpMac` y se envía el paquete tanto a la LAN como a la *honeynet*, esto se hace debido a que se quiere que las tablas ARP de las dos redes (LAN y *honeynet*) estén pobladas para que en el caso de descubrir un atacante y desviarlo

a la *honeynet*, no se pierda la conectividad y el atacante no perciba el desvío de los paquetes hacia la *honeynet*.

Si el paquete recibido es del tipo IP, primero se verifica si el paquete recibido proviene de la *honeynet*, de ser el caso se procede a hacer una segunda verificación; la existencia o no de un registro en el diccionario `dicMacPuerto` permite que hacer con el paquete. La ausencia del registro indica que el paquete está destinado a una equipo de la LAN, y por lo tanto se descarta puesto que no debe existir conectividad entra la LAN y la *honeynet*, si existe el registro en el diccionario, entonces se procede a enviar el paquete únicamente al atacante, permitiendo la conectividad entre estos dos.

En el caso de que el paquete recibido no proviene de la *honeynet*, se verifica si la dirección IP origen se encuentra en el diccionario `dicIpMac`, si no se encuentra en el registro se desvía hacia la *honeynet*, pues se ha establecido que si un *host* no ha enviado ni recibido ningún paquete ARP, no puede establecer comunicación a nivel del protocolo IP, pero, si está en el diccionario y la dirección MAC asociada a la dirección IP del registro son las mismas direcciones de origen del paquete, entonces se lo toma como un paquete legítimo y se lo deja pasar a la LAN, en cualquier otro caso, se agrega la dirección MAC de origen del paquete a la lista de atacantes y se guarda el número de puerto por el que llego en el diccionario `dicMacPuerto` para cuando llegue un paquete dirigido al atacante solo se lo envíe hacia este, finalmente este paquete se desvía a la *honeynet*.

3.2.2.4 Diagrama de flujo del módulo tcp.py

El diagrama de flujo del módulo `tcp.py` se muestra en la Figura 3.6 y en la Figura 3.7.

En este módulo se utilizan los siguientes arreglos:

- `dicSolicitudes.-` Contiene la dirección IP del equipo al que está dirigido la solicitud, en conjunto con el número de solicitudes que haya enviado.
- `dicClientes.-` Diccionario que contiene las direcciones IP de los equipos que son considerados como clientes y el número de solicitudes enviadas por los mismos.

- `lstAtacantes`.- Lista que contiene las direcciones IP de los equipos que han sido detectados como atacantes.

El proceso que se realiza en este módulo es el siguiente:

Inicia cuando se recibe un paquete, lo primero que se hace es verificar si se trata de un paquete TCP, si se trata de cualquier otro tipo de paquetes se revisa si provienen o no de un atacante, y dependiendo de eso se lo deja pasar a la LAN o se lo desvía hacia la *honeyne*t.

En el caso de que se reciba un paquete TCP, entonces primero se verifica si proviene o no de la dirección IP del servidor, de ser así, se lo deja pasar a la LAN. Si el paquete TCP recibido proviene de otro origen entonces se lo somete a un procesamiento más extenso.

En primera instancia se debe diferenciar entre los distintos tipos de paquetes que se envían en un inicio de sesión TCP (procedimiento que se realiza en tres vías como ya se mencionó en la Sección 1.4.1.2), pero solo interesa discriminar entre los paquetes `TCP_SYN` (primera vía de conexión) y el `TCP_ACK` (tercera vía de conexión).

Si el paquete recibido no es del tipo `TCP_SYN`, se procede a comprobar si se trata de un paquete del tipo `TCP_ACK`, en caso de que no lo fuera, se verifica si la dirección IP de origen está o no en la lista de atacantes para proceder a enviar el paquete a la LAN o desviarlo a la *honeynet*.

Si el paquete recibido efectivamente es un `TCP_ACK`, esto indica la confirmación de la conexión y por lo tanto garantiza que no se trata del ataque `TCP_SYN_FLOOD`, en consecuencia, si la dirección IP de origen está en el diccionario de solicitudes se elimina el registro del diccionario, se la agrega al diccionario `dicClientes` y se lo envía a la LAN.

Si la dirección IP de origen está en lista de atacantes, entonces se elimina de la lista de solicitudes y se la agrega al diccionario `dicClientes`.

Pero si ya estaba en el diccionario `dicClientes` se procede a disminuir en uno el número de solicitudes y se envía a la LAN. Si la cuenta llega a cero deja de disminuir

y se envía el paquete a la LAN, se realiza de esta forma para que no disminuya infinitamente la cuenta.

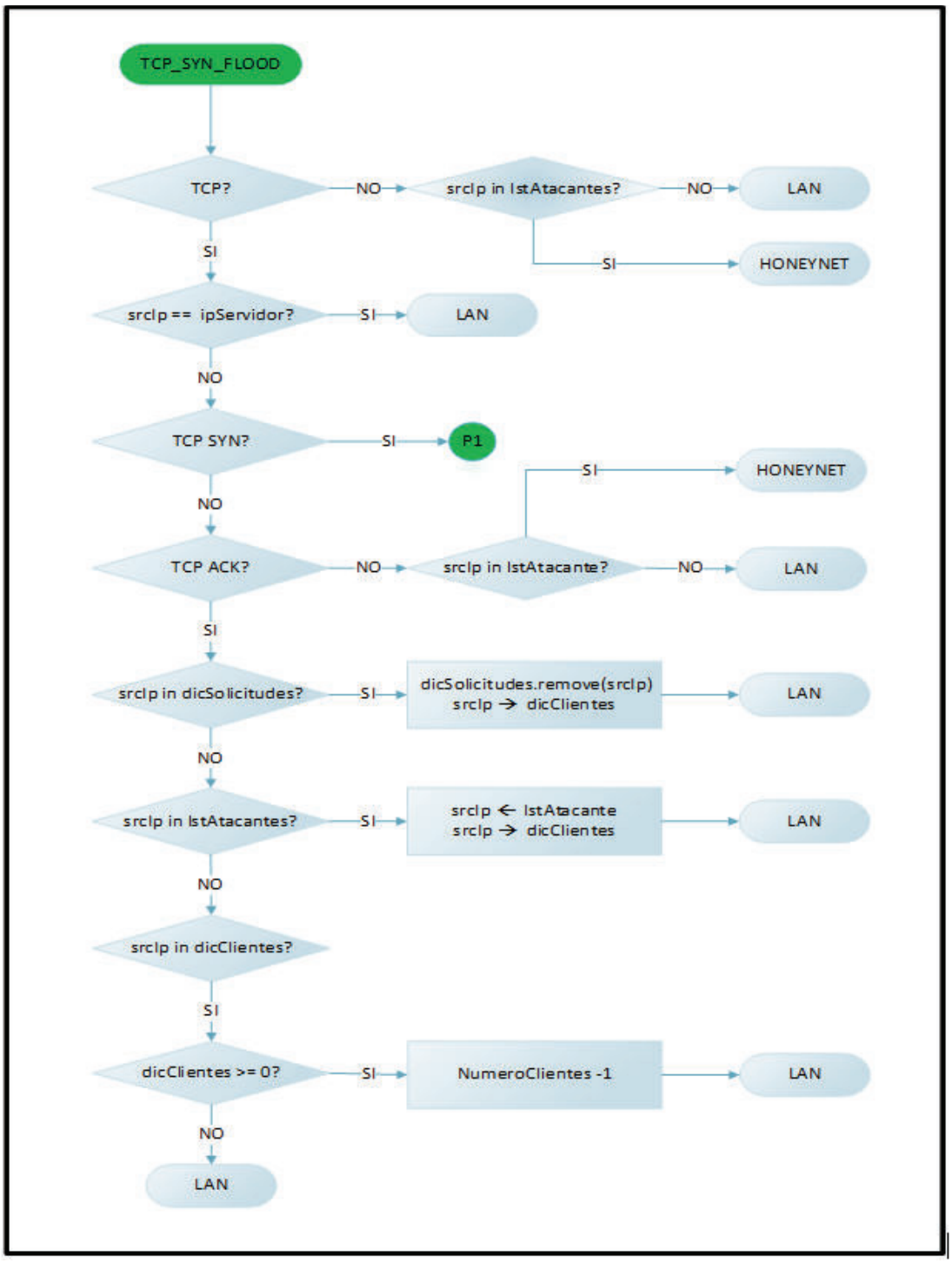

Figura 3.6 Diagrama de flujo del módulo `tcp.py` (Sección Primera)

En el caso de que el paquete recibido corresponda a un `TCP_SYN`, se debe tener mucho cuidado porque podría efectivamente tratarse de un ataque de inundación de paquetes de intento de conexión `TCP_SYN`.

En este caso el procedimiento de verificación, el cual se presenta de forma gráfica en la Figura 3.7, consiste principalmente en verificar si la dirección IP de origen del paquete recibido esta en la lista de atacantes, de ser el caso, el paquete es desviado a la *honeynet*.

Si no se encuentra en la lista de atacantes, se prosigue a verificar si se encuentra en el diccionario `dicSolicitudes`. Si efectivamente se encuentra en este diccionario, entonces se debe comprobar que el número de solicitudes recibidas sea menor al número máximo de solicitudes establecido en el archivo de configuración, si es así, se debe incrementar en uno el número de solicitudes de esa dirección IP y se deja pasar el paquete a la LAN; pero si el número de conexiones que están registradas en el diccionario de solicitudes ha alcanzado el número de solicitudes máximo permitido, entonces se procede a eliminar el registro del diccionario `dicSolicitudes`, y se agrega la dirección IP origen del paquete a la lista de atacantes.

En caso de que la dirección IP de origen, no esté en el diccionario `dicSolicitudes`, entonces se comprueba si está en el diccionario `dicClientes`, si tampoco está en este diccionario, entonces se agrega en el diccionario `dicSolicitudes` indicando que es la primera conexión.

En caso de que si exista el registro de la dirección IP origen del paquete recibido en el diccionario `dicClientes`, entonces se debe verificar que el número de conexiones activas sea menor al número máximo de conexiones permitidas por cada cliente.

Si el número de conexiones es menor al máximo permitido, entonces se incrementa en uno y se lo deja pasar a la LAN, pero si el número de conexiones desde un cliente ha alcanzado el valor máximo, se elimina el registro de este diccionario y se agrega la dirección IP origen del paquete a la lista de atacantes.

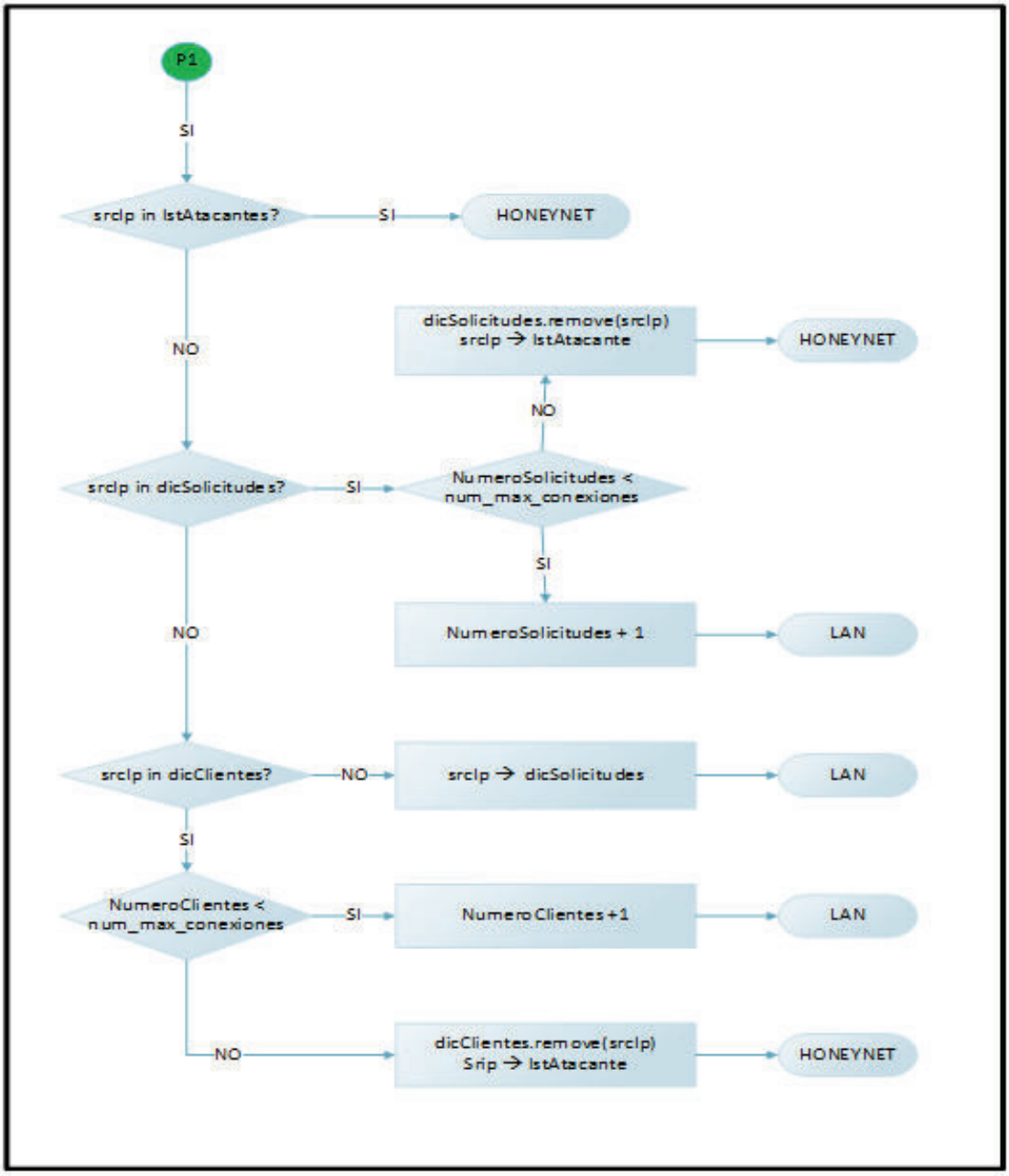

Figura 3.7 Diagrama de flujo del módulo `tcp.py` (Sección Segunda)

De esta manera cuando un ataque de este tipo provenga de una determinada dirección IP, se desvía hacia la *honeynet* todo el tráfico que genere mientras se comporte como atacante, sin embargo si al acabo de un intervalo de tiempo nuevamente actúa como un cliente legítimo, entonces, se elimina de la lista de atacantes y se permite su tráfico nuevamente a hacia la LAN.

3.2.2.5 Diagrama de flujo del módulo https.py

El diagrama de flujo del módulo `https.py` al igual que el del módulo `tcp.py`, se encuentra esquematizado en dos figuras. En la Figura 3.8 se puede observar que a la llegada de un paquete, se verifica en primer lugar si se trata de un paquete TCP, puesto que el protocolo HTTPS se encuentra encapsulado en este, y para que empiece la negociación de conexión de SSL, debe entonces previamente haberse establecido la sesión TCP. Si se trata de cualquier otro tipo de paquete se verifica si proviene de una dirección IP que este en la lista de atacantes para dejarlo pasar a la LAN o desviarlo a la *honeynet*.

En este módulo se utilizan los siguientes arreglos:

- `dicSolicitudes.-` Contiene la dirección IP del equipo al que está dirigido la solicitud y el número de solicitudes enviadas.
- `dicClientes.-` Diccionario que contiene la dirección IP de los equipos que son considerados como clientes y el número de solicitudes enviadas.
- `lstAtacantes.-` Lista que contiene la dirección IP de los equipos que han sido detectados como atacantes.

Luego de esto, si se confirma que se trata de un paquete TCP, entonces se verifica que no se origine en el servidor, si es el caso, entonces se permite pasar el paquete a la LAN.

Si el paquete recibido proviene de un origen diferente al servidor, entonces se debe verificar si se trata de un paquete de inicio de sesión SSL, denominado `HELLO_CLIENT`, o si se trata de un paquete posterior al establecimiento de la conexión denominado `SSL_DATA`, si no se trata de ninguno de estos paquetes, se verifica únicamente que no provengan de una dirección IP que antes ya fue establecida como atacante y por lo tanto se encuentra en la lista de atacantes, de no ser así se deja pasar a la LAN pues puede corresponder a paquetes TCP de otras aplicaciones o inclusive a los paquetes TCP de establecimiento de la sesión TCP previa a la negociación SSL.

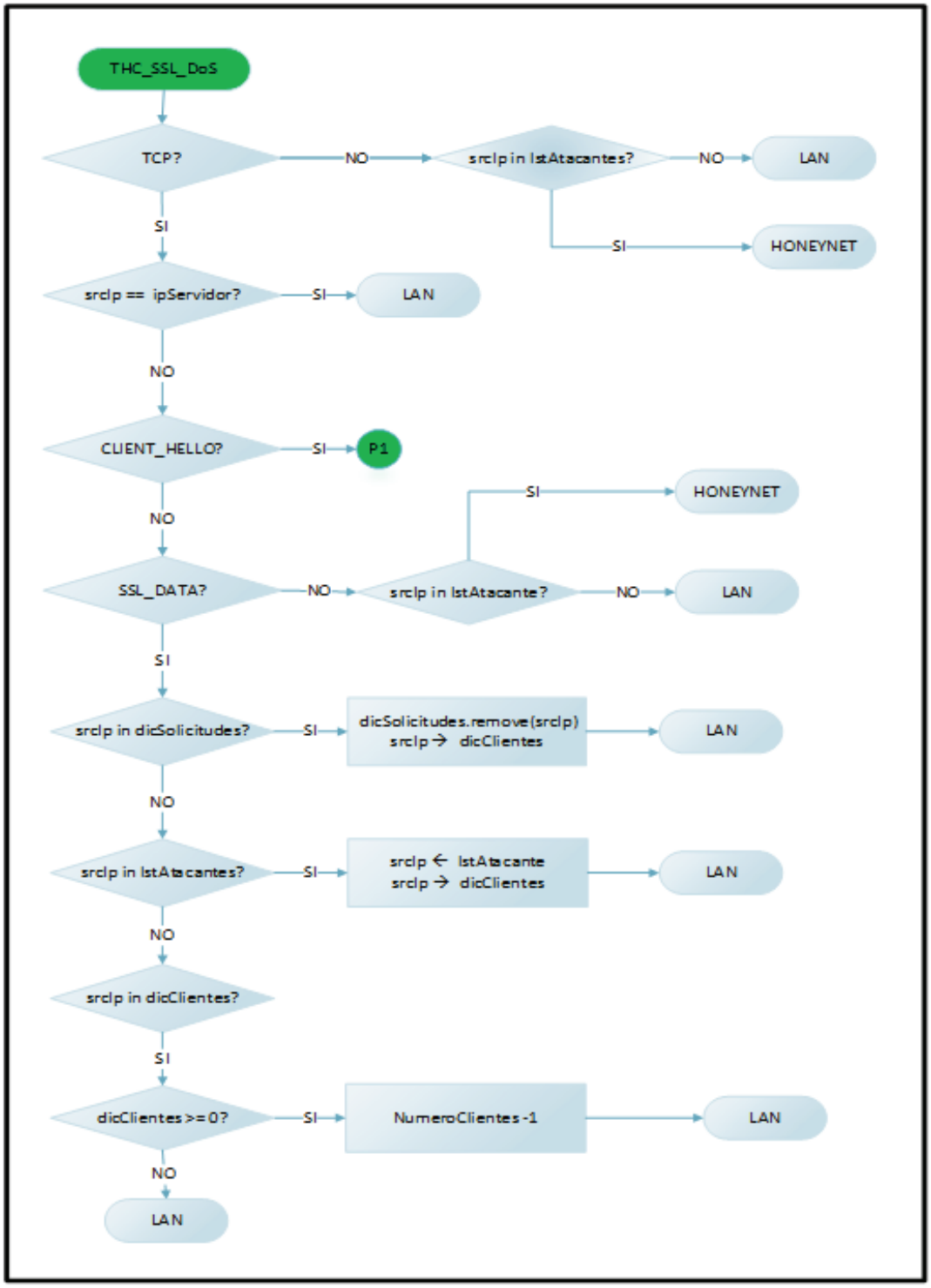

Figura 3.8 Diagrama de flujo del módulo `https.py` (Sección Primera)

Si el paquete recibido es del tipo `SSL_DATA`, esto indica que se trata de un usuario legítimo, puesto que el ataque `THC_SSL_DoS` no trabaja con paquetes de datos,

por lo tanto al recibir este paquete se verifica en primer lugar si la dirección IP de origen está en el diccionario `dicSolicitudes`. De ser así, entonces se elimina esa entrada del diccionario y se agrega una nueva entrada en el diccionario `dicClientes` indicando que se trata de la primera conexión.

En caso de que la dirección IP de origen no se encuentre en el diccionario `dicSolicitudes`, se verifica a continuación si se encuentra en la lista de atacantes, si efectivamente está en esta lista, se procede a eliminar este registro y se lo agrega al diccionario `dicClientes` indicando de la misma manera que se trata de la primera conexión.

Finalmente, si no se encuentra en la lista de atacantes, se verifica que se encuentre en el diccionario `dicClientes`, de ser así, se disminuye en uno cada vez que llegue un paquete de este tipo, siempre que el valor sea mayor o igual a cero, para este caso y el caso contrario se envía a la LAN, de esta forma se evita que la cuenta disminuya infinitamente. Esto ayuda a determinar cuando un cliente supera el número máximo de conexiones permitidas.

Pero en caso de que el paquete recibido, corresponde al tipo `HELLO_CLIENT` de inicio de establecimiento de conexión SSL, entonces se ejecuta el subproceso denominado `P1` mostrado en la Figura 3.9.

En primer lugar, se verifica si la dirección IP de origen está en la lista de atacantes, de ser así, se envía directamente hacia la *honeynet* y se evita un mayor procesamiento.

Sin embargo, en caso de que no se encuentre en la lista de atacantes, entonces, se verifica si se encuentra en el diccionario `dicSolicitudes`.

Si efectivamente se encuentra en dicho diccionario, entonces se verifica si el número de conexiones que se registran en el diccionario de solicitudes, no supera al número de solicitudes máximo permitidas por cliente, si es menor entonces se procede a incrementar dicho número en uno y se deja pasar el paquete a la LAN. Pero en el caso de que el número de solicitudes registradas ya haya alcanzado el número máximo permitido, se procede a eliminar la entrada del diccionario

`dicSolicitudes` y se agrega la dirección IP a la lista de atacantes y a continuación se envía este paquete a la *honeynet*.

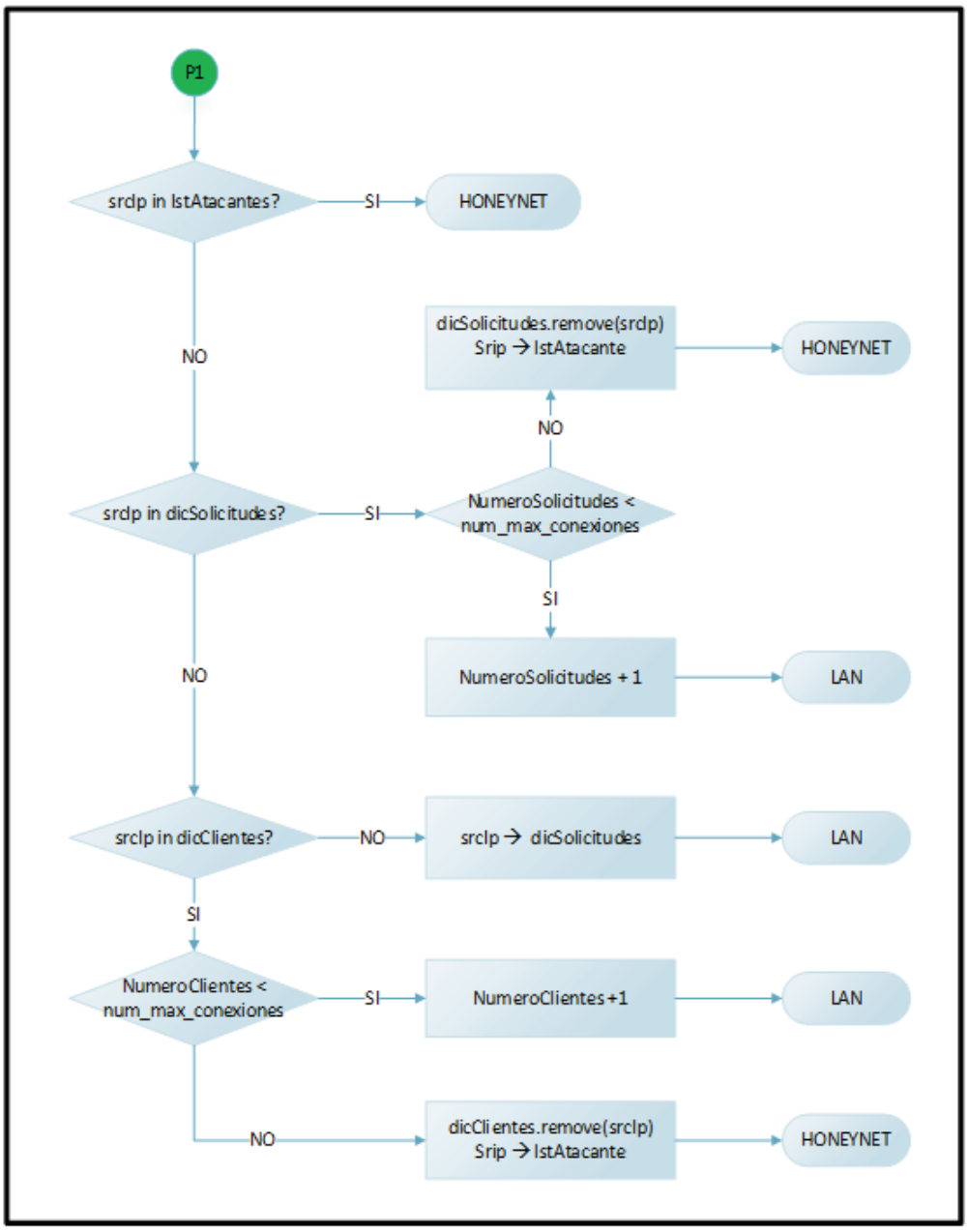

Figura 3.9 Diagrama de flujo del módulo `https.py` (Sección Segunda)

Por otra parte, si la dirección IP de origen tampoco se encontraba en el diccionario `dicSolicitudes`, entonces se procede a verificar si se encuentra en el diccionario `dicClientes`, de ser así, entonces se verifica que el número de

conexiones que tenga cada cliente, no supere al número máximo de conexiones permitidas por cliente.

Si el número de conexiones registradas en el diccionario es menor a dicho número, entonces se procede a incrementar en uno dicho número y se deja pasar el paquete a la LAN, pero en el caso de que el número de conexiones registradas en el diccionario haya alcanzado el número máximo, entonces se procede a eliminar la entrada de este diccionario y se agrega a la lista de atacantes.

Pero si la dirección IP de origen tampoco está en el diccionario `dicClientes`, entonces se agrega la direcciona IP de origen al diccionario `dicSolicitudes` y se inicializa en uno el número de solicitudes de este nuevo cliente.

3.2.2.6 Diagrama de flujo del módulo icmp.py

El diagrama de flujo para la detección del ataque denominado `SMURF`, se muestra en la Figura 3.10, en la cual se muestra que lo primero que se hace es distinguir los tipos de paquete, debido a que este ataque trabaja sobre el protocolo ICMP, cualquier paquete encapsulando sobre otro protocolo se deja pasar de forma transparente hacia la LAN.

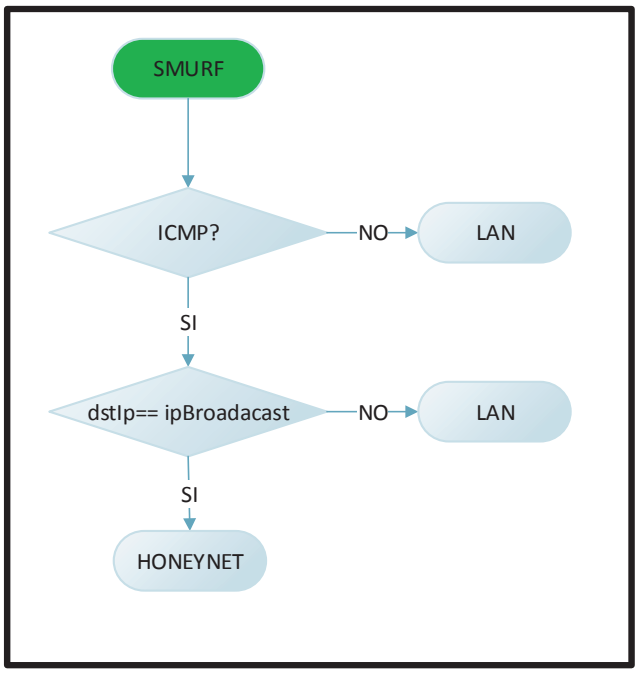

Figura 3.10 Diagrama de flujo del módulo `icmp.py`

En caso de que el paquete recibido sea efectivamente del tipo ICMP, entonces se procede a comprobar que la dirección IP de destino no sea la dirección IP de *broadcast*, en caso de serlo, se desvía el paquete hacia la *honeynet*.

Pero si el paquete recibido tiene una dirección IP destino diferente a la dirección IP de *broadcast*, entonces se deja pasar el paquete de forma transparente hacia la LAN.

3.2.2.7 Diagrama de flujo del módulo udp.py

El diagrama de flujo del módulo `udp.py` se presenta en la Figura 3.11. Inmediatamente al recibir un paquete, se comprueba si la dirección MAC origen se encuentra en la lista de atacantes, de ser el caso, el paquete se desvía hacia la *honeynet* y se evita un mayor procesamiento.

Este módulo utiliza el arreglo `lstAtacantes`, que es una lista que contiene la dirección IP de los equipos que han sido detectados como atacantes.

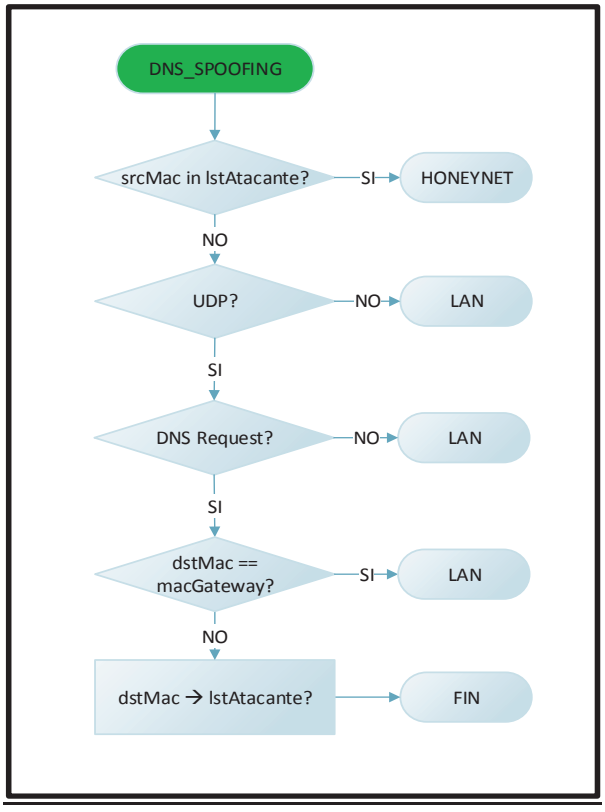

Figura 3.11 Diagrama de flujo del módulo `udp.py`

Sin embargo, si no se encuentra en la lista de atacantes, entonces se comprueba si se trata de un paquete UDP, puesto que el servicio de DNS corre sobre el protocolo UDP.

Si llega cualquier otro protocolo entonces se deja pasar de forma transparente a la LAN, pero si se trata de paquetes UDP, entontes se verifica si se trata de paquetes del tipo `DNS_REQUEST`, cualquier otro tipo de paquete se deja pasar de la misma manera a la LAN sin ninguna restricción, pero si se determina que un paquete corresponde efectivamente al tipo `DNS_REQUEST`, entonces se debe verificar adicionalmente si la dirección MAC de destino corresponde a la dirección MAC del *gateway*, si es así, se deja pasar a la LAN, caso contrario se agrega la dirección IP de destino a la lista de atacantes y se descarta el paquete.

3.2.2.8 Diagrama de flujo del módulo enviar.py

Finalmente, en la Figura 3.12, Figura 3.13 y Figura 3.14, se presenta el diagrama de flujo del último módulo implementado, el módulo `enviar.py`.

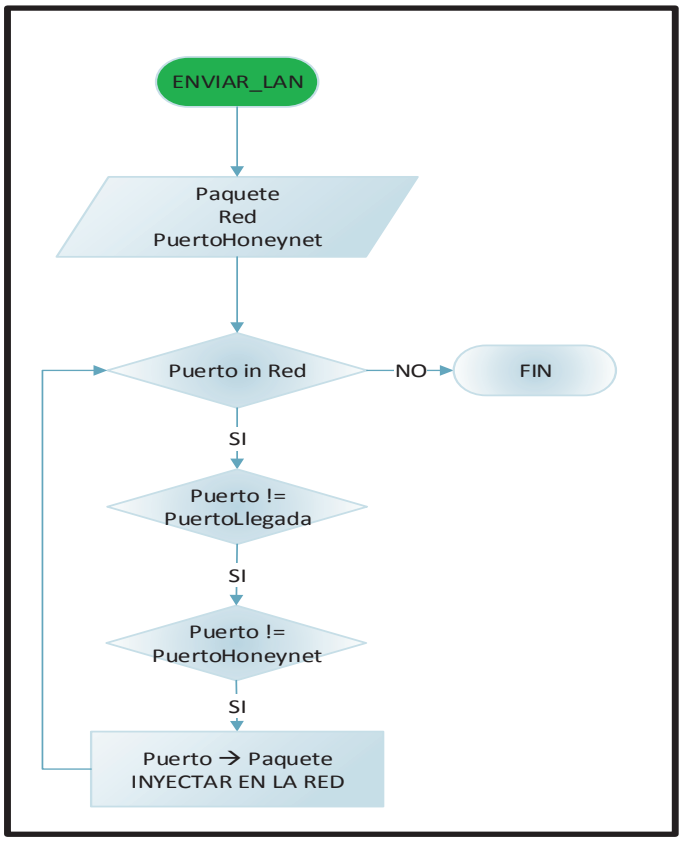

Figura 3.12 Diagrama de flujo del módulo `enviar.py` (método `enviar_lan`)

La Figura 3.12 muestra el algoritmo para enviar un paquete a la LAN, para ello se requiere el paquete a enviar, la red (en ella se incluye la topología, y por tanto se conoce los switches OpenFlow y los puertos que poseen) y el puerto OpenFlow en el que se encuentra conectada la *honeynet*.

Se procede mediante un lazo a enviar cada paquete por cada uno de los puertos OpenFlow del switch. Por cada puerto, se procede a verificar que no sea ni el puerto por el que llego el paquete, ni tampoco el puerto de la *honeynet*, luego se modifica el campo puerto en el paquete y finalmente se lo inyecta en la red.

La Figura 3.13 muestra el envío de un paquete únicamente al puerto de la *honeynet*, para ello se procede a modificar en el paquete el campo puerto con el puerto de la *honeynet* y a continuación se lo inyecta en la red.

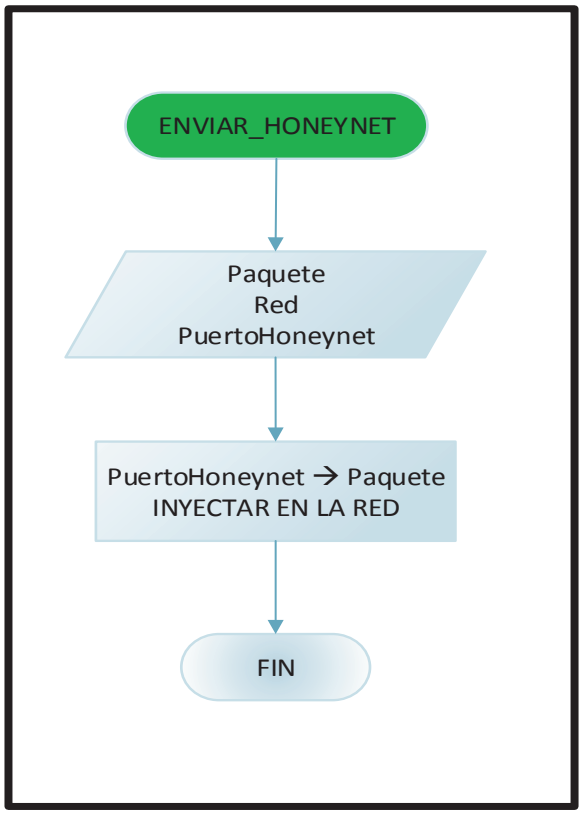

Figura 3.13 Diagrama de flujo del módulo `enviar.py` (método `enviar_honeynet`)

Finalmente, la Figura 3.14 muestra el diagrama para el envío de un paquete a todos los puertos OpenFlow de la red excepto al puerto por el que llego, para ello

se procede a modificar el paquete con el número de puerto por el que se quiere enviar, tantas veces como puertos OpenFlow haya en la red, siempre y cuando este no sea el puerto por el que haya llegado el paquete y se procede a inyectar los paquetes en la red.

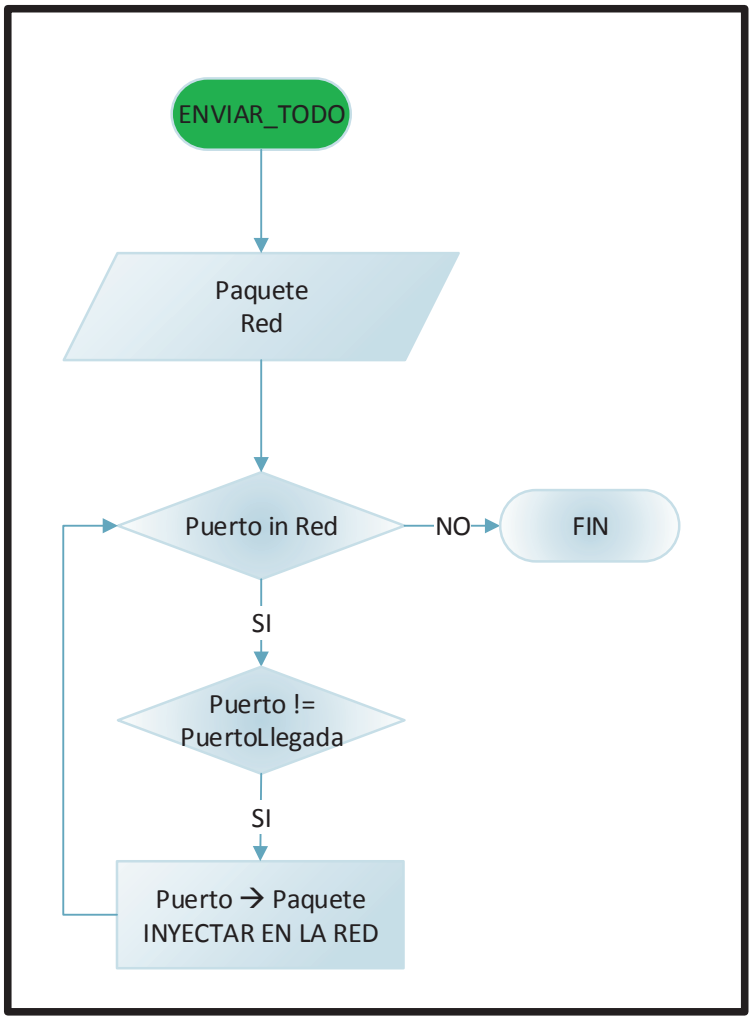

Figura 3.14 Diagrama de flujo del módulo `enviar.py` (método `enviar_todo`)

CAPÍTULO IV

4 IMPLEMENTACIÓN DE LA APLICACIÓN Y PRUEBAS

En este capítulo se presenta el código de la aplicación para el controlador Pyretic que ha sido implementada como solución ante los diferentes ataques informáticos, presentados en los capítulos previos.

Se presenta el código de la aplicación dividida en módulos acompañada de una explicación detallada de los mismos. Adicionalmente, al final se presenta las estadísticas obtenidas de acuerdo a cada ataque informativo para las diferentes pruebas realizadas.

4.1 IMPLEMENTACIÓN DEL MÓDULO CONTROLADOR

El módulo controlador, o modulo principal, es uno de los módulos más complejos, aquí se recibe cada uno de los paquetes que le lleguen al switch SDN, y es el encargado de inicializar las diferentes listas, diccionarios, variables globales, entre otros.

El módulo principal está también encargado de enviar dichos paquetes a los diferentes módulos de procesamiento que se requieran para una configuración específica. También recibe la respuesta del módulo de procesamiento al que envío el paquete y luego lo inyecta en la red utilizando el método apropiado del módulo de envío.

El Segmento de código 4.1 muestra la primera parte del módulo `controlador.py`. Luego de la descripción del módulo, se importan los módulos: `arp.py`, `ip.py`, `tcp.py`, `udp.py`, `icmp.py` y `https.py`, posteriormente se importan todas las clases y métodos de las dos librerías más importantes de Pyretic, y finalmente la librería que permite leer archivos de configuración.

```
################################################################################
#                          ESCUELA POLITECNICA NACIONAL                        #
# -----------------------------------------------------------------------------#
# Tema: Aplicación para el controlador Pyretic                                 #
# Subtema: Módulo controlador.py                                               #
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)         #
#  Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)          #
```

```
# Fecha: Lunes  20 de  Octubre de 2014                                    #
##########################################################################

      import arp
      import ip
      import tcp
      import udp
      import icmp
      import https
      import enviar
      from pyretic.lib.corelib import *
      from pyretic.lib.query import *
      from ConfigParser import ConfigParser
```

Segmento de código 4.1 Módulo `controlador.py` (importación de librerías y módulos)

El Segmento de código 4.2 muestra la declaración de la única clase del módulo `controlador.py` denominada `ControladorHoneynet`, las dos líneas posteriores a la declaración de la clase, sirven para crear una instancia de la clase que permite leer archivos de configuración y para indicarle cual archivo de configuración se deberá leer. Luego, las cuatro líneas siguientes son las declaraciones de los diccionarios y listas que se utilizan en el módulo `arp.py`. Las tres líneas siguientes son las declaraciones de los diccionarios y de la lista que se usan en el módulo `ip.py`. Las tres líneas siguientes, en cambio, indican la declaración de los diccionarios y lista que se utilizan en el módulo `tcp.py`. Las siguientes tres líneas son las declaraciones de los diccionarios y listas que se utilizan en el módulo `https.py`. Finalmente se tiene la declaración de la lista que se utiliza en el módulo `udp.py`.

Las últimas seis líneas nos indican la carga de valores de diferentes variables utilizadas en los distintos módulos obtenidos del archivo de configuración.

```
class ControladorHoneynet(DynamicPolicy):
    config = ConfigParser()
    config.read("honeynet.cfg")

    dicSolicitudesA = {}
    dicMacIpA = {}
    dicMacPuertoA = {}
    lstAtacantesA = []

    dicIpMacI = {}
    dicMacPuertoI = {}
    lstMacAtacanteI = []
```

```
    dicSolicitudesT = {}
    dicClientesT = {}
    lstAtacantesT = []

    dicSolicitudesS = {}
    dicClientesS = {}
    lstAtacantesS = []

    ListaAtacantesU = []

    puertoHoneynet = config.getint("PUERTOS", "puertoHoneynet")
    ipServidor = config.get("IPS","ipServidor")
    ipBroadcast = config.get("IPS","ipBroadcast")
    macGateway = config.get("MACS","macGateway")
    num_max_conexiones =
config.getint("CONEXIONES","numeroConexiones")
    proceso = config.getint("PROCESOS","proceso")
```

Segmento de código 4.2 Módulo `controlador.py` (declaración de variables globales y lectura del archivo de configuración)

En el Segmento de código 4.3 se muestra la declaración del constructor de la clase y del método `set_network`. En el constructor se indica que cada paquete que llegue sea procesado en esta clase, es decir se llama a la clase que indica que cada paquete que se reciba en el controlador se lo procese en este módulo.

El método `set_network` en cambio permite determinar la topología de la red que se tiene.

```
 def __init__(self):
     self.query = packets()
     self.query.register_callback(self.paquete)
     self.network = None
     super(ControladorHoneynet,self).__init__(self.query)


   def set_network(self, network):
     self.network = network
```

Segmento de código 4.3 Módulo *`controlador.py`* (declaración del constructor de la clase y del método *`set_network`*)

Cabe resaltar, que el código de los segmentos indicados hasta el momento se ejecuta una sola vez al iniciar el controlador, luego de ello únicamente el método que se presenta en el Segmento de código 4.4 se ejecuta cada vez que llegue un paquete al switch.

En el Segmento de código 4.4 se muestra la declaración del método `paquete`, este método se ejecuta cada vez que se recibe un paquete en el controlador. Cada vez que se recibe un paquete, se obtiene la dirección MAC destino, útil para el envío de paquetes a un destino específico, luego se inicializa la variable `respuesta` como una cadena de caracteres vacía, luego se verifica cual es el valor de la variable `proceso`, la cual fue cargada desde el archivo de configuración al iniciarse el controlador.

```
def paquete(self,pkt ):

        dstmac = pkt ['dstmac']
        respuesta = ""
        if proceso == 0:

            respuesta = arp.arp_spoofing(pkt, dicSolicitudesA,
dicMacIpA, dicMacPuertoA, ListaAtacantesA)
            if respuesta == "LAN":
                  respuesta = ip.ip_spoofing(pkt,  dicIpMacI,
dicMacPuertoI, lstMacAtacanteI, puertoHoneynet)
                  if respuesta == "LAN":
                        respuesta = tcp.tcp_syn_flood(pkt,
lstAtacantesT, dicSolicitudesT, dicClientesT, IP(ipServidor),
num_max_permitido)
                        if respuesta == "LAN":
                              respuesta = udp.dns_spoofing(pkt,
lstAtacantesU, macGateway)
                              if respuesta == "LAN":
                                    respuesta = icmp.smurf(pkt,
ipBroadcast)
                                    if respuesta == "LAN":
                                          respuesta =
https.thc_ssl_dos(pkt, lstAtacantesS, dicSolicitudesS, dicClientesS,
IP(ipServidor), num_max_permitido)
```

Segmento de código 4.4 Módulo `controlador.py` (declaración del método `paquete` - Parte I)

La primera verificación es para saber si `proceso` es igual a cero, en cuyo caso, como se indicó en la Sección 3.2.1, se deben ejecutar secuencialmente los módulos que sean necesarios dependiendo del tipo de paquete que se reciba. Entonces se

llama al método `arp_spoofing` declarado en el módulo `arp.py` y se le pasa los argumentos necesarios, esto se detallará más adelante cuando se revise el código de cada módulo. Todos los métodos de procesamiento de los diferentes módulos, devuelven una variable *string* como respuesta, en la que indican que se debe hacer con el paquete, por ejemplo, cuando devuelve "`LAN`", significa que es un paquete legítimo que debe dejarse pasar a la LAN o que simplemente se debe dejar pasar pues es un paquete que no se considere peligroso. Para aclarar esto considere el siguiente ejemplo: si la respuesta del método `arp_spoofing` es "`LAN`", significa que no es un paquete peligroso o que el paquete recibido es de un tipo diferente (TCP, UDP, etc.) y que por lo tanto no representa ningún peligro de ser un ataque de tipo `ARP_SPOOFING`, puesto que dicho ataque funciona únicamente con paquetes del tipo ARP.

De la misma manera, si la respuesta obtenida de este método es "`LAN`", eso indica que no se está ejecutando un ataque de tipo `IP_SPOOFING`, por lo tanto se procede a llamar al siguiente módulo de procesamiento. De esta manera se ejecuta de forma secuencial uno a uno los métodos necesarios para la detección de los diferentes ataques, declarados en cada uno de los módulos.

En el Segmento de código 4.5 se muestran las otras opciones de procesamiento, dependiendo del valor que tenga la variable `proceso`.

```
   elif proceso == 1:
            respuesta = arp.arp_spoofing(pkt, dicSolicitudesA,
dicMacIpA, dicMacPuertoA, ListaAtacantesA)

      elif proceso == 2:
      respuesta = ip.ip_spoofing(pkt,  dicIpMacI, dicMacPuertoI,
lstMacAtacanteI, puertoHoneynet)

      elif proceso == 3:
            respuesta = tcp.tcp_syn_flood(pkt, lstAtacantesT,
dicSolicitudesT, dicClientesT, ipServidor, num_max_permitido

      elif proceso == 4:
            respuesta = udp.dns_spoofing(pkt, lstAtacantesU,
macGateway)

      elif proceso == 5:
      respuesta = icmp.smurf(pkt, ipBroadcast)

      elif proceso == 6:
```

```
            respuesta = https.thc_ssl_dos(pkt, lstAtacantesS,
dicSolicitudesS, dicClientesS, ipServidor, num_max_permitido
```

Segmento de código 4.5 Módulo `controlador.py` (declaración del método `paquete` - Parte II)

Cuando esta variable toma valores de 1 a 6, se ejecuta únicamente un módulo de procesamiento, los valores asociados a los módulos se indicaron en la Sección 3.2.1. Cuando se llama a un determinado método, se le pasa los argumentos necesarios en las que figuran las listas y diccionarios declarados al inicio del módulo, como se presentó en el Segmento de código 4.2.

Por lo tanto al finalizar cualquier modulo que se haya procesado, este devuelve la variable `respuesta` que indica que se debe hacer con el paquete.

En el Segmento de código 4.6 se presenta la llamada el método de envío que se requiera dependiendo de la respuesta obtenida: Si la respuesta obtenida del módulo de procesamiento es ”`LAN`”, entonces se procede a llamar al método `enviar_lan`, para enviar el paquete a cada uno de los puertos OpenFlow que haya en la red, excepto el puerto por el que llego y el puerto de la *honeynet*.

```
         if respuesta == "LAN":
            enviar.enviar_lan(pkt, red, puertoHoneynet)
        elif respuesta == "HONEYNET":
            enviar.enviar_honeynet(pkt, red, puertoHoneynet)
        elif respuesta == "TODO":
            enviar.enviar_todo(pkt, red)
        elif respuesta == "ATACANTE":
            enviar.enviar_honeynet(pkt, red, dicMacPuerto[dstmac])

def main():
      return ControladorHoneynet()
```

Segmento de código 4.6 Módulo `controlador.py` (declaración del método `paquete` - Parte III)

Si la respuesta es ”`HONEYNET`” o ”`ATACANTE`”, se procede a llamar al método `enviar_honeynet`, este método permite enviar únicamente a un puerto

específico, ya sea el puerto de la *honeynet* o por el puerto en el que se encuentre el atacante.

Finalmente si la respuesta es "`TODO`", entonces se procede a enviar el paquete por todos los puertos OpenFlow de la red, sin importar que se el puerto por el que llego, el puerto de la *honeynet* e inclusive sin importar que sea el puerto en el que se encuentre algún atacante.

Las dos últimas líneas muestran la definición del método principal `main( )`. Este método se encarga de llamar al método `ControladorHoneynet`, que es el método que se acaba de revisar.

4.2 IMPLEMENTACIÓN DEL MÓDULO ARP

Al igual que en el caso anterior, el modulo `arp.py` se inicia con un encabezado y la importación de librerías, como se muestra en el Segmento de código 4.7. La primera librería permite manejar colecciones, útil para el manejo de listas y diccionarios y las siguientes tres importaciones son las librerías propias de Pyretic.

```
#########################################################################
###
#                          ESCUELA POLITECNICA NACIONAL
#
# ----------------------------------------------------------------------
--#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo arp.py
#
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)
#
#  Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre de 2014
#
#########################################################################
###
import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.7 Módulo `arp.py` (Sección I)

Una vez importadas las librearías necesarias, se procede a declarar el método que procesará los paquetes recibidos, este método se denomina `ARP_SPOOFING`, y recibe como parámetros de entrada los siguientes:

- `pkt`: paquete encapsulado en el protocolo OpenFlow.
- `dicSolicitudes`: Diccionario que guarda pares clave-valor, en donde la clave es la dirección IP a la que se le realiza una solicitud ARP (dirección IP destino), y cuyo valor es un número entero que especifica el número de paquetes de solicitud que se han enviado y que están pendientes de contestar.
- `dicMacIp`: Diccionario que permite almacenar pares clave-valor, en donde la clave en este caso es la dirección MAC de origen de paquetes que correspondan a solicitudes ARP y cuyo valor es la dirección IP origen de los mismo paquetes.
- `dicMacPuerto`: Diccionario que permite almacenar los pares clave-valor, a clave corresponde a las direcciones MAC de los paquetes que han sido clasificados como provenientes de un atacante y cuyo valor, es el número de puerto por el que haya llegado el paquete.
- `lstAtacantes`: Lista en la que se almacenan las direcciones MAC de los paquetes que han sido clasificados como provenientes de un atacante.

En el Segmento de código 4.8, se muestra la declaración del método y el código del algoritmo mostrado en la Sección 3.2.2.2 representado en la Figura 3.4.

En primer lugar, se extraen del paquete recibido los campos:

- `ethtype`: correspondiente al tipo de paquete a nivel de capa 2 según el modelo de referencia ISO/OSI.
- `srcmac`: Dirección MAC de origen del paquete.
- `dstmac`: Dirección MAC destino del paquete.

Posteriormente se declara la variable `respuesta` como una cadena de caracteres vacía, que será la que retorna el método.

En primera instancia se extraen únicamente ciertos campos que se los pueden encontrar en todo tipo de paquete.

A continuación, se verifica el tipo de paquete recibido, si es del tipo ARP (`ethtype` igual a 2054 en decimal), entonces se lo procesa, si es cualquier otro tipo de paquete entonces se lo envía a donde corresponda.

Si se determina que se trata de un paquete ARP, entonces se procede a extraer los campos:

- `inport`: Número de puerto OpenFlow por el que llego el paquete.
- `protocolo`: Determina el tipo específico de paquete ARP, es decir si se trata de un `ARP_REQUEST` o de un `ARP_REPLAY`.
- `dstip`: Dirección IP de destino del paquete
- `srcip`: Dirección IP de origen del paquete.

Luego de lo cual, se verifica que tipo de paquete se recibió. Si el campo `protocolo` es "`1`", entonces se trata de un paquete del tipo `ARP_REQUEST`, si el campo `protocolo` es "`2`", entonces se trata de un paquete del tipo `ARP_REPLAY`.

En el Segmento de código 4.8 se muestra el código que se ejecuta cuando efectivamente el campo `protocolo` tiene un valor igual a "`1`".

```
def arp_spoofing(pkt,dicSolicitudes, dicMacIp,  dicMacPuerto,
lstAtacantes):

      tipoPkt = pkt['ethtype']
      srcmac = pkt['srcmac']
      dstmac = pkt['dstmac']
      respuesta = ""

      if tipoPkt == 2054:
            inport = pkt['inport']
            protocolo = pkt['protocol']
            dstip  = pkt['dstip']
            srcip = pkt['srcip']

            if protocolo == 1:
                  dicMacIp[srcmac]= srcip
                  if dicSolicitudes.has_key(dstip):
                        dicSolicitudes[dstip] = dicSolicitudes[dstip]+1
                        respuesta = "LAN"
                  else:
                        dicSolicitudes[dstip]= 1
                        respuesta = "LAN"
```

Segmento de código 4.8 Módulo `arp.py` (Sección II)

Si el campo `protocolo` es “`1`”, se trata de una solicitud ARP, entonces se guarda en el diccionario `dicMacIp` el par {dirección MAC, dirección IP}.

Cuando se recibe una solicitud ARP, se toma como verdaderos los valores de los campos de la solicitud pues este ataque se basa únicamente en el envío de respuestas ARP, por lo tanto, se considera como auténticos todos los datos contenidos en el diccionario `dicMacIp`.

Luego de agregarlos o actualizarlos en el diccionario, se verifica si ya existió previamente alguna solicitud ARP a la dirección IP a la que va dirigida la solicitud recibida.

Si no existe, entonces se la agrega inicializada en uno, si ya existía, entonces únicamente se incrementa en uno el número de solicitudes pendientes de respuesta.

Sin embargo cuando se recibe un paquete ARP del tipo `ARP_REPLY`, entonces se ejecuta el Segmento de código 4.9.

```
 elif protocolo == 2:
      if dicMacIp.has_key(srcmac):
            if dicMacIp[srcmac] == srcip:
                  respuesta = "LAN"
                  if srcmac in lstAtacantes:
                        lstAtacantes.remove(srcmac)
                  else:
                        del dicMacIp[srcmac]
                        respuesta = "HONEYNET"
            else:
                  if dicSolicitudes.has_key(srcip):
                        if  dicSolicitudes[srcip] >= 0:
                              dicSolicitudes[srcip] =
dicSolicitudes[srcip] - 1
                              respuesta = "LAN"
                        else:
                              lstAtacantes.append(srcmac)
                              dicMacPuerto[srcmac]= inport
                               respuesta = "HONEYNET"
                  else:
                        lstAtacantes.append(srcmac)
                        dicMacPuerto[srcmac]= inport
                        respuesta = "HONEYNET"
      else:
            respuesta = "LAN"
```

Segmento de código 4.9 Módulo `arp.py` (Sección III)

Cuando se recibe un paquete cuyo campo `protocolo` es igual a "`2`", entonces se trata de una respuesta `ARP_REPLY`, por lo que se procede a verificar si la dirección MAC origen del paquete recibido, se encuentra en el diccionario `dicMacIp` (es decir se verifica si se realizó alguna solicitud desde esa dirección MAC), de ser así, se comprueba si la dirección IP asociada a la dirección MAC del registro del diccionario, es igual a la dirección IP de origen del paquete recibido.

En caso afirmativo, se envía el paquete a la LAN, y si la dirección MAC desde donde provino el paquete está en la lista de atacantes `lstAtacantes`, entonces se la elimina de esta lista.

Pero si la dirección IP del registro es diferente a la dirección IP origen del paquete recibido, entonces se elimina el registro del diccionario `dicMacIp`, y se desvía el paquete hacia la *honeynet*. La eliminación del registro del diccionario se ejecuta con el fin de dar apertura a la posibilidad de que el servidor DHCP le asigno una nueva dirección IP, en cuyo caso debería necesariamente realizar peticiones ARP y se volvería a agregar un nuevo registro en el diccionario `dicMacIp`, en el que se asocia la dirección MAC origen con la nueva dirección IP del host.

Cuando se recibe un paquete desde una dirección MAC que no ha realizado previamente una solicitud ARP, lo que implica que no se encuentra ningún registro asociado a dicha dirección MAC en el diccionario `dicMacIp`, entonces se verifica si existen ya solicitudes ARP dirigidas a la misma direcciones IP a donde se dirige el paquete recibido. Y adicionalmente se debe verificar que el número de solicitudes sea mayor a cero, se deja pasar el paquete recibido hacia la LAN y se disminuye en uno el número de solicitudes pendientes de respuesta asociadas a una determinada dirección IP.

En caso de que no se han recibido solicitudes, o en caso de que el número de solicitudes recibidas disminuyó a cero, en los dos casos se procede a agregar la dirección MAC origen a la lista de atacantes `lstAtacantes`, pues no es posible que alguien conteste a una solicitud que no ha sido enviada.

Adicionalmente, antes de desviar el paquete hacia la h*oneyne*t, se guarda el número de puerto OpenFlow asociado a la dirección MAC origen del paquete para saber en qué puerto está conectado el presunto atacante.

Cualquier otro tipo de paquete se deja pasar a la LAN sin ninguna restricción.

Finalmente, en el Segmento de código 4.10 se muestran las acciones tomadas, cuando se recibe un paquete que no sea del tipo ARP.

```
    else:
          if dstmac in lstAtacantes:
                respuesta = "ATACANTE"
          elif srcmac in lstAtacantes:
                respuesta = "HONEYNET"
          else:
                respuesta = "LAN"

     return respuesta
```

Segmento de código 4.10 Módulo `arp.py` (Sección IV)

Se verifica si la dirección MAC de destino se encuentra en la lista de atacantes `lstAtacantes` (es decir el paquete se dirige a un presunto atacante), entonces se envía el paquete únicamente por el puerto OpenFlow en el que se encuentra el atacante.
Si no es el caso, se verifica si la dirección MAC de origen se encuentra en la lista de atacantes `lstAtacantes`, (lo que implica que el paquete recibido proviene de una atacante), y entonces se envía el paquete hacia la *honeynet*. Cualquier otro paquete que no cumpla ninguna de estas condiciones, se deja pasar a la LAN.

4.3 IMPLEMENTACIÓN DEL MÓDULO IP

El modulo `ip.py` se inicia con un encabezado y la importación de librerías, como se muestra en el Segmento de código 4.11, la primera librería permite manejar colecciones, y las siguientes tres importaciones son las librerías propias de Pyretic.

```
###################################################################
###
#                        ESCUELA POLITECNICA NACIONAL
#
# ---------------------------------------------------------------
--#
```

```
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo ip.py
#
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)
#
#   Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre de 2014
#
###################################################################
###

import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.11 Módulo `ip.py` (Sección I)

Una vez importadas las librearías necesarias, se procede a declarar el método que procesará los paquetes recibidos, el Segmento de código 4.12 muestra el método denominado `IP_SPOOFING`, el cual recibe como parámetros de entrada los siguientes:

- `pkt`: Paquete encapsulado en el protocolo OpenFlow.
- `dicIpMac`: Diccionario que permite almacenar pares clave-valor, en donde la clave es la dirección IP de origen de los paquetes recibidos y cuyo valor es la dirección MAC origen de los mismos paquetes.
- `dicMacPuerto`: Diccionario que permite almacenar los pares clave-valor, en donde la clave son las direcciones MAC de los paquetes que han sido clasificados como provenientes de un atacante y cuyo valor, es el número de puerto por el que llegó el paquete.
- `lstAtacante`: Lista en la que se almacenan las direcciones MAC de los paquetes que han sido clasificados como provenientes de un atacante.
- `puertoHoneynet`: Número de puerto OpenFlow del switch en el que está conectada la *honeynet*.

Luego de la declaración del método, se extraen ciertos valores de los campos del paquete, como son: `ethtype`, `srcip`, `srcmac`, y `dstmac`. Y posteriormente se declara la variable `respuesta`. Ya explicados en la Sección anterior.

```
def ip_spoofing(pkt, dicIpMac, dicMacPuerto, lstAtacante,
puertoHoneynet):
        tipoPkt = pkt['ethtype']
        srcip = pkt['srcip']
        dstmac = pkt ['dstmac']
        srcmac = pkt['srcmac']
        respuesta = ""
```

Segmento de código 4.12 Módulo `ip.py` (Sección II)

Una vez extraídas las variables necesarias, se verídica en primera instancia si la dirección MAC origen del paquete recibido está en la lista de atacantes `lstAtacantes`, como se muestra en el Segmento de código 4.13 de ser el caso se desvía inmediatamente el paquete a la *honeynet* evitando así un mayor procesamiento.

Si no se encuentra en esa lista entonces se verifica si el paquete recibido es del tipo ARP, en cuyo caso se guarda el par clave-valor {dirección IP origen, dirección MAC origen}, en el diccionario `dicIpMac` y se devuelve la variable respuesta igual a ”`TODO`”, lo que indica que se debe enviar el paquete a todos los puertos OpenFlow, incluido el de la *honeynet*.

```
    if srcmac in lstAtacante:
        respuesta = "HONEYNET"
    else:
        if tipoPkt == 2054:
            dicIpMac[srcip]= srcmac
            return  "TODO"
        elif tipoPkt == 2048:
            inport = pkt['inport']
            if inport == puertoHoneynet:
                if dicMacPuerto.has_key(dstmac):
                    return "ATACANTE"
                else:
                    return "FIN"
            else:
                if dicIpMac.has_key(srcip):
                    if srcmac == dicIpMac[srcip]:
                        respuesta = "LAN"
                    else :
                        lstAtacante.append(srcmac)
                        dicMacPuerto[srcmac] = inport
                        respuesta = "HONEYNET"
                else:
                    respuesta = "LAN"
```

```
            else:
             respuesta = "LAN"

    return respuesta
```

Segmento de código 4.13 Módulo `ip.py` (Sección III)

Si no se trata de un paquete ARP, entonces se verifica si se trata de un paquete del tipo IP (`ethtype` igual a 2048 en decimal), en cuyo caso se procede a extraer una variable adicional del paquete, el número de puerto OpenFlow por el que llego el paquete (`inport`), para verificar si es el mismo puerto que el de la *honeynet* (verifica si el paquete proviene de la *honeynet*), de ser así, entonces se procede a descartarlo a menos que hay una entrada en el diccionario `dicMacPuerto`, en la que figure la dirección IP destino del paquete y tenga asociado un numero de puerto, es decir el paquete no es descartado si se dirige a un atacante, si existe el registro entonces se devuelve la variable respuesta igual a ”`ATACNTE`”.

Por otra parte, si el paquete no proviene de la *honeynet*, entonces se comprueba si existe un registro en el diccionario `dicIpMac` cuya clave sea la dirección IP origen del paquete recibido y adicionalmente si existe tal registro que la dirección MAC asociada a dicha dirección IP del registro, sea igual a la dirección MAC origen del paquete recibido.

De darse tal situación, entonces el paquete se deja pasar a la LAN. Pero si las direcciones MAC comparadas, son diferentes, entonces se agrega la dirección MAC origen del paquete recibido a la lista de atacantes `lstAtacantes`, se guarda el número de puerto OpenFlow por el que llego el paquete asociado a la clave dirección MAC en el diccionario `dicMacPuerto` y se procede a desviar el paquete hacia la *honeynet*.

En cualquier otro caso, se deja pasar los paquetes a la LAN.

4.4 IMPLEMENTACIÓN DEL MÓDULO TCP

El modulo `tcp.py` se inicia con un encabezado del módulo y la importación de librerías, como se muestra en el Segmento de código 4.14, la primera librería permite manejar colecciones, y las siguientes tres importaciones son las librerías propias de Pyretic.

```
#########################################################################
###
#                          ESCUELA POLITECNICA NACIONAL
#
# ----------------------------------------------------------------------
--#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo tcp.py
#
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)
#
#   Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre
#########################################################################
###

import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.14 Módulo `tcp.py` (sección I)

Una vez importadas las librerías necesarias, se procede a declarar el método que procesará los paquetes recibidos. El Segmento de código 4.15 muestra el método denominado `tcp_syn_flood`, el cual tiene como parámetros de entrada los siguientes:

- `pkt`: Paquete encapsulado en el protocolo OpenFlow.
- `lstAtacantes`: Lista en la que se almacenan las direcciones MAC de los paquetes que hayan sido clasificados como provenientes de un atacante.
- `dicSolicitudes`: Diccionario que permite almacenar pares clave-valor, en donde la clave es la dirección IP de origen de los paquetes recibidos y cuyo valor es el número de paquetes del tipo `TCP_SYN`.
- `dicClientes`: Diccionario que permite almacenar los pares clave-valor, en donde la clave son las dirección IP origen de los paquetes que han sido clasificados como provenientes de un cliente legítimo y el valor número de conexiones establecidas o completadas en el momento actual.
- `ipServidor`: Dirección IP del servidor (blanco del ataque).
- `num_max_permitido`: Número máximo de solicitudes y/o conexiones simultáneas permitido desde un mismo origen.

Luego de la declaración del método se extraen ciertos valores de los campos del paquete, como: `ethtype`, `protocol`, y `srcip`. Explicado en la sección 4.2.

Debido a que Pyretic únicamente determina ciertas políticas y variables específicas contenidas en los encabezados de los paquetes (no todas), se necesita obtener ciertos campos de las cabeceras de los paquetes, generalmente a nivel de las capa 3 y 4 según el modelo de referencia ISO/OSI.

El método que permite extraer campos de las capas 3 y 4 se muestra en el Segmento de código 4.19, y se explicara a detalle más adelante. Haciendo uso de dicho método se procede a obtener las banderas del encabezado TCP para poder determinar el tipo de paquete que se está procesando.

Finalmente, en el Segmento de código 4.15, luego de haber obtenido las variables necesarias, se procede a verificar si se trata de un paquete del tipo TCP para ello se debe cumplir que la variable `tipoPkt` sea de tipo IP, y que la variable `protocolo` sea igual a "`6`" (lo que indica que se trata de TCP), se der el caso, se verifica adicionalmente si la dirección IP origen del paquete corresponde a la dirección IP del servidor (es decir el paquete proviene del servidor), de ser así, se deja pasar a la LAN pues el servidor no puede ser el atacante.

```
def tcp_syn_flood(pkt, lstAtacantes, dicSolicitudes, dicClientes,
ipServidor, num_max_permitido):
      tipoPkt = pkt['ethtype']
      protocolo = pkt['protocol']
      srcip = pkt['srcip']

      tcp_flags = payload(pkt, 94, 96)
      respuesta = ""

      if tipoPkt == 2048 and protocolo == 6:
            if ipServidor == srcip:
                  respuesta = "LAN"
```

Segmento de código 4.15 Módulo `tcp.py` (Sección II)

Cuando ha llegado un paquete del tipo TCP, y no proviene del servidor, entonces se procede a verificar el tipo específico de paquete TCP para poder realizar las acciones pertinentes.

En el Segmento de código 4.16 se muestra el algoritmo que se ejecuta cuando se trata de un paquete TCP del tipo `TCP_SYN`.

Si las banderas TCP del paquete recibido son iguales a “02”, se confirma que se trata de un paquete TCP_SYN (primera vía de conexión, indica un requerimiento de establecimiento de conexión), en cuyo caso, en primer lugar se verifica si la dirección IP origen del paquete está en la lista de atacantes lstAtacantes, de ser así, se devuelve al módulo controlador la variable respuesta igual a ”HONEYNET”, para que se envíe este paquete únicamente a la *honeynet*.

En segundo lugar y al no tratarse de un paquete proveniente de un atacante, entonces se procede a verificar si en el diccionario dicSolicitudes, se encuentra un registro cuya clave es la dirección IP origen del paquete recibido; en tal situación, se debe verificar si el número de solicitudes asociadas a una determinada dirección IP, no sobrepase el número máximo de solicitudes permitido.

Si es así, se incrementa en uno el número de solicitudes provenientes de la dirección IP origen del paquete recibido y se devuelve la variable respuesta con el valor ”LAN”, con el fin de que se deje pasar de forma transparente hacia la LAN.

Pero si se alcanzó el número máximo de solicitudes permitidas para la dirección IP origen desde donde proviene el paquete, entonces se elimina del diccionario dicSolicitudes el registro asociado a esta dirección IP, se agrega esta dirección IP a la lista de atacantes y se procede a devolver la variable respuesta con el valor igual a ”HONEYNET”.

En tercer lugar, y luego de haber verificado que el emisor del paquete (dirección IP origen) no ha sido clasificado ni cómo atacante, ni como solicitante, entonces se procede a verificar si se trata de un cliente.

```
if str(tcp_flags) == "02":
      if srcip in lstAtacantes:
        respuesta  = "HONEYNET"
      else:
        if dicSolicitudes.has_key(srcip):
          if dicSolicitudes[srcip] < num_max_permitido:
                dicSolicitudes[srcip] = dicSolicitudes[srcip] + 1
                respuesta  = "LAN"
          else:
                del dicSolicitudes[srcip]
                lstAtacantes.append(srcip)
```

```
                respuesta  = "HONEYNET"
        else:
          if dicClientes.has_key(srcip):
                if dicClientes[srcip] < num_max_permitido:
                      dicClientes[srcip] = dicClientes[srcip] + 1
                      respuesta  = "LAN"
                else:
                      del dicClientes[srcip]
                      lstAtacantes.append(srcip)
                      respuesta  = "HONEYNET"
          else:
                dicSolicitudes[srcip] = 1
                respuesta  = "LAN"
```

Segmento de código 4.16 Módulo `tcp.py` (sección III)

Si se trata de un cliente, adicionalmente se debe verificar que el número de conexiones del mismo no supere el número máximo permitido. Si no se supera este límite, entonces se incrementa en uno el número de conexiones que posee y se devuelve la variable `respuesta` igual a ”`LAN`”. En caso de que se alcance el número máximo de conexiones remitidas por cliente, se elimina la entrada asociada a este cliente del diccionario de clientes, se lo agrega a la lista de atacantes y se devuelve la variable `respuesta` igual a ”`HONEYNET`”.

Finalmente, se el emisor del paquete (dirección IP origen) no se encuentra en la lista de atacantes, en el diccionario de solicitudes ni en el diccionario de clientes, quiere decir que es el primer paquete que se recibe desde un determinado origen, en tal caso se lo agrega al diccionario de solicitudes, con el número de conexión inicializado en uno.

El Segmento de código 4.17 muestra el código que se ejecuta cuando se recibe un paquete TCP del tipo `TCP_ACK` (tercera vía de conexión - confirmación del establecimiento de la conexión).

```
else:
      if str(tcp_flags) == "10":
        if dicSolicitudes.has_key(srcip):
              del dicSolicitudes[srcip]
              dicClientes[srcip] = 1
              respuesta  = "LAN"
        else:
              if srcip in lstAtacantes:
```

```
                        lstAtacantes.remove(srcip)
                        dicClientes[srcip] = 1
                        respuesta  = "LAN"
                  else:
                        if dicClientes.has_key(srcip):
                              if dicClientes[srcip] >= 0:
                                    dicClientes[srcip] =
dicClientes[srcip] - 1
                              respuesta  = "LAN"
          else:
            if srcip in lstAtacantes:
                  respuesta  = "HONEYNET"
            else:
                  respuesta  = "LAN"
```

Segmento de código 4.17 Módulo `tcp.py` (Sección IV)

Si el paquete recibido corresponde al tipo `TCP_ACK`, entonces se verifica primero que el diccionario de solicitudes tenga una entrada con la clave igual al dirección IP de origen del paquete recibido, si existe, entonces se la elimina y se agrega una nueva entrada en el diccionario de clientes, indicando que se trata de la primera conexión y se devuelve la variable `respuesta` igual a "`LAN`".

Si no existe el registro en el diccionario de solicitudes entonces se verifica si esa dirección IP origen está en la lista de atacantes, en cuyo caso, se elimina de la lista de acatantes y se agrega al diccionario de clientes de la misma manera indicando que es la primera conexión establecida.

En caso de que la dirección IP origen del paquete recibido no este ni en el diccionario de solicitudes, ni en la lista de atacantes, entonces se verifica si está en el diccionario de clientes, si es así, entonces se disminuye en uno cada vez que lleguen paquetes de este tipo, hasta llegar a cero y se procede a devolver la variable `respuesta` con el valor de "`LAN`".

Si el paquete recibido no corresponde al tipo `TCP_SYN` y tampoco al tipo `TCP_ACK`, entonces para cualquier otro tipo de paquete (de negociaciones establecimiento de conexión o de datos) se verifica que no esté en la lista de atacantes y se lo deja pasar de forma transparente hacia la LAN, para ello se devuelve la variable `respuesta` igual a "`LAN`", pero si proviene de un atacante, entonces se la desvía a la *honeynet*, para ello se devuelve la variable `respuesta` con el valor de "`HONEYNET`", tal como se observa en el Segmento de código 4.18.

```
          else:
            if (srcip in lstAtacantes):
              respuesta  = "HONEYNET"
            else:
              respuesta = "LAN"
    return respuesta
```

Segmento de código 4.18 Módulo `tcp.py` (Sección V)

El segundo método que posee este módulo, es el método que permite obtener ciertos campos adicionales del paquete recibido, el cual está encapsulado en OpenFlow. El método se presenta en el Segmento de código 4.19.

```
def payload(pkt,num1,num2):
          of_payload_code = pkt['raw']
          of_payload = of_payload_code.encode("hex")
          bandera = of_payload[num1:num2]
          return bandera
```

Segmento de código 4.19 Módulo `tcp.py` (Sección VI)

Este método tiene como parámetros de entrada:

- `pkt`: Paquete de datos recibido en el controlador encapsulado en OpenFlow
- `num1`: Número que determina la posición inicial del campo o variable del encabezado que se desea recuperar.
- `num2`: Número que determina la posición final del campo o variable del encabezado que se desea recuperar.

Primero se extrae y asigna a la variable `of_payload_code` el *payload* del paquete OpenFlow codificado.

Luego, el *payload* OpenFlow codificado, se lo decodifica a hexadecimal obteniendo una cadena de caracteres hexadecimales y se la asigna a la variable `of_payload`.

Finalmente, se asigna a la variable bandera la porción de la cadena delimitada por los números `num1 y num2`. Se retorna dicho valor.

4.5 IMPLEMENTACIÓN DEL MÓDULO UDP

En el Segmento de código 4.20, se muestra el encabezado del módulo y la importación de las librerías necesarias, la primera importación es la librería que permite trabajar con colecciones y las tres siguientes son las librerías del núcleo de Pyretic.

```
#########################################################################
###
#                          ESCUELA POLITECNICA NACIONAL
#
# ----------------------------------------------------------------------
--#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo udp.py
#
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)
#
#   Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre de 2014
#
#########################################################################
###

import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.20 Módulo `udp.py` (sección I)

Una vez importadas las librerías, como se muestra en el Segmento de código 4.21 se procede a declarar el método que se ejecuta para la detección de este ataque, dicho método se llama `DNS_SPOOFING` y recibe los siguientes parámetros de entrada como argumentos:

- `pkt`: Paquete encapsulado en el protocolo OpenFlow.
- `lstAtacantes`: Lista en la que se almacenan las direcciones MAC destino de los paquetes que hayan sido clasificados como dirigidos a un atacante.
- `macGateway`: Dirección MAC del *gateway* de la red.

Una vez declarado el método, entonces se procede a extraer ciertos campos de las cabeceras del paquete, como: `ethtype`, `protocol`, `dstmac`, y `srcmac`. Y posteriormente se declara la variable `respuesta`. Explicados en la Sección 4.2.

A continuación, se extraen las banderas del paquete DNS mediante el método `payload`, que se revisará más adelante.

Una vez que se han extraído las banderas, se verifica que el paquete recibido no provenga de un atacante, es decir que la dirección MAC origen del paquete recibido no se encuentre en la lista de atacantes, de ser así entonces se devuelve la variable `respuesta` con el valor de "`HONEYNET`".

Pero si el paquete no proviene de un atacante, entonces se verifica que se trate de un paquete UDP, encapsulado en IP (solicitudes y respuestas DNS). Si corresponde a este tipo de paquete, entonces se verifica que el paquete recibido corresponda al tipo `DNS_REQUEST` (petición DNS).

Debido a que para este ataque se debió anteriormente haber realizado un ataque `ARP_SPOOFING`, es necesario también comprobar que cualquier paquete `DNS_REQUEST` que se reciba, este dirigido a la dirección MAC del *gateway*, en tal caso, se dejará pasar de forma transparente a la LAN, para lo cual se devuelve la variable `respuesta` con el valor de "`LAN`".

Sin embargo, si la solicitud DNS recibida tiene una dirección MAC diferente a la del *gateway* (dirigida a otro usuario), entonces se toma como atacante el *host* a quien va dirigido el paquete y por lo tanto se procede a agregar su dirección MAC en la lista de atacantes y posteriormente se descarta el paquete, para ello se devuelve la variable `respuesta` con el valor "`FIN`".

Cualquier otro tipo de paquete que no es `DNS_REQUEST`, se deja pasar de forma transparente hacia la LAN y para ello se devuelve la variable `respuesta` con el valor "`LAN`".

```
def dns_spoofing(pkt, lstAtacantes, macGateway):
  tipoPkt = pkt['ethtype']
  protocolo = pkt['protocol']
  dstmac = pkt['dstmac']
  srcmac = pkt['srcmac']
```

```
      respuesta = ""

      dns_flags = payload(pkt,88,92)

      if srcmac in lstAtacantes:
            respuesta = "HONEYNET"
      else:
            if tipoPkt == 2048 and protocolo == 17:
                    if (dns_flags == '0100'):
                      if dstmac ==  macGateway:
                        respuesta = "LAN"
                      else:
                        lstAtacantes.append(dstmac)
                        respuesta = "FIN"
                    else:
                      respuesta = "LAN"
            else:
                    respuesta = "LAN"
      return respuesta
```

Segmento de código 4.21 Módulo `udp.py` (Sección II)

Finalmente, el modulo posee un método similar al mostrado en el Segmento de código 4.19, para la extracción de campos de las cabeceras delos diferentes protocolos que encapsulan al paquete.

4.6 IMPLEMENTACIÓN DEL MÓDULO ICMP

En primera instancia, como se muestra en el Segmento de código 4.22 se tiene la cabecera del módulo seguido de la importación de las librerías necesarias en este módulo. La primera librería se utiliza para el manejo de colecciones y las tres siguientes son la importación de las librerías propias del controlador.

```
#########################################################################
###
#                          ESCUELA POLITECNICA NACIONAL
#
# ----------------------------------------------------------------------
--#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo icmp.py
#
# Programador: Naranjo Villota Jose Luis  (joseluisnaranjo@hotmail.es)
#
#   Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre de 2014
#
```

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

import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.22 Módulo `icmp.py` (sección I)

Luego de realizarse las importaciones de las librerías necesarias, se declara el método `smurf` que se encarga de detectar este tipo de ataque:

Como se observa en el Segmento de código 4.23. Este método tiene como parámetros de entrada los siguientes:

- `pkt`: Paquete encapsulado en el protocolo OpenFlow.
- `ipBroadcast`: Dirección IP de difusión de la red de trabajo.

```
def smurf(pkt, ipBroadcast):
  tipoPkt = pkt['ethtype']
  protocolo = pkt['protocol']
  dstip = pkt['dstip']

  if tipoPkt == 2048 and protocolo == 1:
          if (dstip == ipBroadcast):
            respuesta = "HONEYNET"
                 else:
            respuesta = "LAN"
  else:
          respuesta = "LAN"
  return respuesta
```

Segmento de código 4.23 Módulo `icmp.py` (Sección II)

Y, a continuación se extraen del paquete ciertas variables declaradas en el núcleo de Pyretic, como son: `ethtype`, `protocol`, y `dstip`. Explicados en la Sección 4.2.

Luego de lo cual se verifica el tipo de paquete que se ha recibido, para ello se comprueba si se trata de paquetes IP y que encapsule al protocolo ICMP (`protocol` = ”1”), cualquier otro tipo de paquete se deja pasar de forma transparente a la LAN.

Si se trata de paquetes ICMP, se verifica si van dirigidos a la dirección IP de *broadcast*, y si es así se los desvía hacia la *honeynet*, para lo cual se devuelve la variable `respuesta` con el valor de "`HONEYNET`".

4.7 IMPLEMENTACIÓN DEL MÓDULO HTTPS

En primer lugar, como se muestra en el Segmento de código 4.24. Se muestra el encabezado del módulo y la importación de las librerías necesarias en el módulo, al igual que en los casos anteriores.

```
#####################################################################
##
#                     ESCUELA POLITECNICA NACIONAL
#
# ------------------------------------------------------------------
-#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo https.py
#
# Programador: Naranjo Villota Jose Luis (joseluisnaranjo@hotmail.es)
#
#  Manzano Barrionuevo Andres Michael (andres_manzano017@hotmail.com)
#
# Fecha: Lunes  20 de  Octubre de 2014
#
#####################################################################
##

import collections
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.24 Módulo `https.py` (Sección I)

- Posteriormente, como se muestra en el Segmento de código 4.25, se declara el método denominado `thc_ssl_dos`, que se ejecuta para detectar este tipo de ataque informático, y cuyos parámetros de entrada son los siguientes:`pkt`: Paquete encapsulado en el protocolo OpenFlow.
- `lstAtacantes`: Lista en la que se almacenan las direcciones IP de los paquetes que han sido clasificados como provenientes de un atacante.
- `dicSolicitudes`: Diccionario que permite almacenar pares clave-valor, en donde la clave es la dirección IP de origen de los paquetes recibidos y

cuyo valor es el número de paquetes del tipo `HELLO_CLIENT` (inicio de negociación de conexión).

- `dicClientes`: Diccionario que permite almacenar los pares clave-valor, en el cual la clave es la dirección IP origen de los paquetes que hayan sido clasificados como provenientes de un cliente legítimo y el número de conexiones establecidas o completadas en el momento actual.
- `ipServidor`: Dirección IP del servidor (blanco del ataque).
- `num_max_conexiones`: Número máximo de solicitudes y/o conexiones simultáneas permitido desde un mismo origen.

```
def thc_ssl_dos( pkt, lstAtacantes, dicSolicitudes, dicClientes
,ipServidor, num_max_conexion):

      tipoPkt = pkt['ethtype']
      protocolo = pkt['protocol']
      srcip  = pkt['srcip']

      ssl_flags1= payload(pkt, 118, 120)
      ssl_flags2 = payload(pkt, 142, 144)
      ssl_dato1 = payload(pkt, 108, 110)
      ssl_dato2 = payload(pkt, 132, 134)

      respuesta = ""

      if tipoPkt == 2048 and protocolo == 6:
            if ipServidor == srcip:
                  respuesta = "LAN"
```

Segmento de código 4.25 Módulo `https.py` (Sección II)

Luego se extraen ciertos valores de los campos del paquete como son: `ethtype`, `protocol`, y `srcip`. Y posteriormente se declara la variable `respuesta`. Explicados en la Sección 4.2.

Posteriormente, se extraen las banderas de SSL que indican si se trata de un paquete del tipo `HELLO_CLIENT` o si se trata de un paquete de datos. El tratamiento es diferente dependiendo del sistema operativo, por ejemplo en Linux, la cabecera de capa 2 posee ciertos bytes adicionales, y por lo tanto la ubicación de la bandera no se encuentra en la misma posición que en el caso de un paquete generado en Windows.

Se procede a verificar si se trata de un paquete del tipo TCP, para ello se debe cumplir que la variable `tipoPkt` sea igual a 2048 (protocolo IP), y que la variable `protocolo` sea igual a 6 (lo que indica que se trata de TCP), esto con el fin de filtrar únicamente los paquetes que podrían encapsular al protocolo SSL. Luego, se verifica adicionalmente y si la dirección IP origen del paquete corresponde a la dirección IP del servidor (es decir el paquete proviene del servidor), de ser así, se deja pasar a la LAN pues el servidor no puede ser el atacante.

Si las banderas SSL del paquete recibido son iguales a "`01`", entonces se confirma que se trata de un paquete `HELLO_CLIENT` (indica un requerimiento de establecimiento de conexión), en cuyo caso, en primer lugar se verifica si la dirección IP origen del paquete está en la lista de atacantes, de ser así, se devuelve al módulo controlador la variable `respuesta` con el valor igual "`HONEYNET`", este desvíe el paquete hacia la *honeynet*.

En segundo lugar se procede a verificar si en el diccionario `dicSolicitudes` se encuentra un registro cuya clave es la dirección IP origen del paquete recibido, en tal situación, se debe verificar si el número de solicitudes asociadas a esta dirección IP no sobrepasa el número máximo de solicitudes permitido.

Si es así, se incrementa en uno el número de solicitudes provenientes de la dirección IP origen del paquete recibido y se devuelve la variable `respuesta` con el valor "`LAN`", con el fin de que el paquete se envíe hacia la LAN.

Pero si se alcanzó el número máximo de solicitudes permitidas para la dirección IP origen desde donde proviene el paquete, entonces se elimina del diccionario `dicSolicitudes` el registro de dicha dirección IP, y se agrega esta dirección IP a la lista de atacantes y se procede a devolver la variable `respuesta` con el valor de "`HONEYNET`", como se observa en el Segmento de código 4.26.

```
else:
   if (str(ssl_flags1) == "01") or (str(ssl_flags2) == "01"):
         if srcip in lstAtacantes:
               respuesta  = "HONEYNET"
         else:
               if dicSolicitudes.has_key(srcip):
```

```
                        if dicSolicitudes[srcip] < num_max_conexiones:
                              dicSolicitudes[srcip] =
dicSolicitudes[srcip] + 1
                              respuesta  = "LAN"
                        else:
                              del dicSolicitudes[srcip]
                              lstAtacantes.append(srcip)
                              respuesta  = "HONEYNET"
                  else:
                        if dicClientes.has_key(srcip):
                              if dicClientes[srcip] <
num_max_conexiones:
                                    dicClientes[srcip] =
dicClientes[srcip] + 1
                                    respuesta  = "LAN"
                              else:
                                    del dicClientes[srcip]
                                    lstAtacantes.append(srcip)
                                    respuesta  = "HONEYNET"
                        else:
                              dicSolicitudes[srcip] = 1
                              respuesta  = "LAN"
```

Segmento de código 4.26 Módulo `https.py` (sección III)

En tercer lugar, y luego de haber verificado que el emisor del paquete (dirección IP origen) no ha sido clasificado ni como atacante, ni como solicitante, se procede a verificar si se trata de un cliente.

Si se trata de un cliente, adicionalmente se debe verificar que el número de conexiones del mismo no supere el número máximo permitido. Si no se supera este límite, se incrementa en uno el número de conexiones que posee y se devuelve la variable `respuesta` igual a "`LAN`". En caso de que se alcance el número máximo de conexiones remitidas por cliente, se elimina la entrada asociada a este cliente del diccionario de clientes, se lo agrega a la lista de atacantes y se devuelve la variable `respuesta` igual a "`HONEYNET`".

Finalmente, se el emisor del paquete (dirección IP origen) no se encuentra en la lista de atacantes, ni en el diccionario de solicitudes, ni en el diccionario de clientes, quiere decir que es el primer paquete que se recibe proveniente de un determinado origen, en tal caso se lo agrega al diccionario de solicitudes con su contador inicializado en uno, y se devuelve la variable `respuesta` con el valor "`LAN`".

El Segmento de código 4.27 muestra el código que se ejecuta cuando se recibe un paquete SSL de datos (lo que significa que la negociación de parámetros SSL ha sido completada de forma exitosa).

```
else:
      if (str(ssl_dato1) == "17") or (str(ssl_dato2) == "17"):
            if dicSolicitudes.has_key(srcip):
                  del dicSolicitudes[srcip]
                  dicClientes[srcip] = 1
                  respuesta  = "LAN"
            else:
                  if srcip in lstAtacantes:
                        lstAtacantes.remove(srcip)
                        dicClientes[srcip] = 1
                        respuesta  = "LAN"
                  else:
                       if dicClientes.has_key(srcip):
                            if dicClientes[srcip] >= 0:
                                 dicClientes[srcip] =
dicClientes[srcip] - 1
                            respuesta  = "LAN"
      else:
            if srcip in lstAtacantes:
                  respuesta  = "HONEYNET"
            else:
                  respuesta  = "LAN"
```

Segmento de código 4.27 Módulo `https.py` (Sección IV)

Si el paquete recibido corresponde a un paquete SSL de datos (banderas de datos = “`17`”), se verifica primero que el diccionario de solicitudes tenga una entrada con la clave igual a la dirección IP de origen del paquete recibido, si existe, entonces se la elimina de ese diccionario y se agrega una nueva entrada en el diccionario de clientes, indicando que se trata de la primera conexión y se devuelve la variable `respuesta` igual a “`LAN`”.

Si no existe el registro en el diccionario de solicitudes se verifica si esa dirección IP origen está en la lista de atacantes, en cuyo caso, se elimina de la lista de acatantes y se agrega al diccionario de clientes indicando que es la primera conexión establecida.

En caso de que la dirección IP origen del paquete recibido no este ni el diccionario de solicitudes, ni en la lista de atacantes, entonces se verifica si está en el diccionario de clientes, si es así, entonces se disminuye en uno cada vez que llegue

un paquete de este tipo, hasta llegar a cero y a continuación se procede a devolver la variable `respuesta` con el valor de "`LAN`".

Si el paquete recibido no corresponde al tipo `HELLO_CLIENT` y tampoco al tipo `SSL_DATA`, para cualquier otro tipo de paquete que llegue, se verifica que no esté en la lista de atacantes y se lo deja pasar de forma transparente hacia la LAN, para ello se devuelve la variable `respuesta` igual a "`LAN`", pero si proviene de un atacante, entonces se la desvía a la *honeynet*, tal como se observa a continuación en el Segmento de código 4.28.

```
          else:
            if (srcip in lstAtacantes):
              respuesta  = "HONEYNET"
             else:
              respuesta = "LAN"
    return respuesta
```

Segmento de código 4.28 Módulo `https.py` (Sección V)

Al final del módulo, se encuentra implementado el método `payload` para obtener ciertos campos del paquete OpenFlow, dicha implementación se muestra en el Segmento de código 4.19, junto con una explicación detallada del mismo.

4.8 IMPLEMENTACIÓN DEL MÓDULO ENVIAR

Finalmente, el último módulo implementado es el módulo de envío, en el cual se implementan tres métodos para inyectar los paquetes en la red, en los puertos correctos.

En el Segmento de código 4.29 se muestra un encabezado y la importación de las librerías necesarias, en este caso, únicamente son las librerías propias de Pyretic.

```
################################################################################
#
#                        ESCUELA POLITECNICA NACIONAL
#
# ------------------------------------------------------------------------------
#
# Tema: Aplicación para el controlador Pyretic
#
# Subtema: Módulo enviar.py
#
```

```
# Programador: Naranjo Villota Jose Luis
(joseluisnaranjo@hotmail.es)#
#   Manzano Barrionuevo Andres Michael
(andres_manzano017@hotmail.com)#
# Fecha: Lunes  20 de  Octubre de 2014
#
#####################################################################
#

from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
```

Segmento de código 4.29 Módulo `enviar.py` (sección I)

En el Segmento de código 4.30, se muestra la implementación del primer método denominado `enviar_lan`, cuyos parámetros de entrada, son:

- `paquete`: Paquete OpenFlow que se procesó y está listo para devolverse a la red.
- `network`: Es una instancia de la clase *network*, que incorpora la topología de la red OpenFlow, incluyendo dispositivos y numero de puertos de los mismos.
- `puertoHoneynet`: puerto OpenFlow en donde se encuentra conectada a la *honeynet*.

Se extraen del paquete, los siguientes campos:

- `switch`: equipo OpenFlow al que debe ser enviado el paquete (en este caso es el único existente en la red).
- `puerto`: Numero de puerto OpenFlow del switch por el que llego.

Se empieza a enviar copias del paquete por cada uno de los puertos que constan en la variable *network*, excepto por el puerto en el que se encuentra conectada a la *honeynet* y el puerto por el que llego. De esta manera se enviaría a todos los dispositivos de la LAN.

Este método es comúnmente utilizado, pues es el que permite enviar los paquetes a la LAN y hacer que esta se comporte de forma normal, cuando no se tiene que desviar paquetes hacia la *honeynet*.

```
def enviar_lan(paquete, network, puertoHoneynet):
    switch = paquete['switch']
    inport = paquete['inport']

    for port in network.topology.egress_locations() -
{Location(switch, inport)} - {Location(switch, puertoHoneynet)}:
        puerto = port.port_no
        paquete = paquete.modify(outport=puerto)
        network.inject_packet(paquete)
        print "Paquete enviado a la LAN!!..."
```

Segmento de código 4.30 Módulo `enviar.py` (sección II)

El método presentado en el Segmento de código 4.31, permite enviar únicamente a un puerto específico de la red SDN, generalmente este método se utiliza cuando se quiere desviar un paquete hacia la *honeynet* o únicamente hacía un puerto especifico, este método recibe como argumento de entrada, el paquete, la red y el puerto de salida.

```
def enviar_honeynet(paquete, network, puerto):
    paquete = paquete.modify(outport=puerto)
    network.inject_packet(paquete)
```

Segmento de código 4.31 Módulo `enviar.py` (Sección III)

Finalmente, como se muestra en el Segmento de código 4.32 el último método de este módulo, se denomina `enviar_todo`, y recibe como parámetros de entrada únicamente:

- `paquete`: Paquete OpenFlow que se procesó y está listo para devolverse a la red.
- `network`: Es una instancia de la clase *network*, que incorpora la topología de la red OpenFlow, incluyendo dispositivos y numero de puertos de los mismos.

Luego, se extraen del paquete las siguientes variables:

- `switch`: equipo OpenFlow al que debe ser enviado el paquete (en este caso es el único existente en la red).
- `puerto`: Numero de puerto OpenFlow del switch por el que llego.

```
def enviar_todo(paquete,network):
    switch = paquete['switch']
    inport = paquete['inport']

    for port in network.topology.egress_locations() -
{Location(switch, inport)}:
        puerto = port.port_no
        paquete = paquete.modify(outport=puerto)
        network.inject_packet(paquete)
```

Segmento de código 4.32 Módulo `enviar.py` (sección IV)

Luego de extraer las variables necesarias, se procede a enviar por cada uno de los puertos de la red OpenFlow copias del paquete, excepto por el puerto por el que llego, esto incluye tanto puertos de la LAN como de la *honeynet*.

4.9 ESTADÍSTICAS OBTENIDAS DEL ATAQUE ARP SPOOFING

Luego de realizar las pruebas de funcionamiento y corregido algunos *bugs*[56] en la aplicación, se han obtenido las siguientes estadísticas para el ataque `ARP_SPOOFING`.

Para la captura de información, se han utilizado herramientas como Wireshark[57] y Colasoft Capsa[58], y la topología utilizada para las pruebas realizadas se muestra a continuación en la Figura 4.1.

Los equipos utilizados son los equipos 2, 3 y 4.

El equipo 2 posee la dirección IP 192.168.0.2 y la dirección MAC 00:00:00:00:00:02 (equipo víctima).

El equipo 3 es una réplica exacta del equipo 2 (honeynet).

---

[56] *Bug*, es un error o fallo en un programa de computador o sistema de software que desencadena un resultado indeseado.
[57] Wireshark es un analizador de protocolos utilizado para solucionar problemas en redes de comunicaciones.
[58] Colasoft Capsa es analizador de red, útil tanto para redes LAN como WLAN, permite realiza captura de paquetes en tiempo real, monitoreo de red 24x7, análisis de protocolo avanzado, decodificación de paquetes en profundidad, etc.

El equipo 4 es el atacante, posee la dirección IP 192.168.0.3 y la dirección MAC 94:de:80:7c:69:7e.

El router (equipo cuya identidad será suplantada por el atacante) posee la dirección IP 192.168.0.60 y cuya dirección MAC es 00:1c:c0:ef:13:f9.

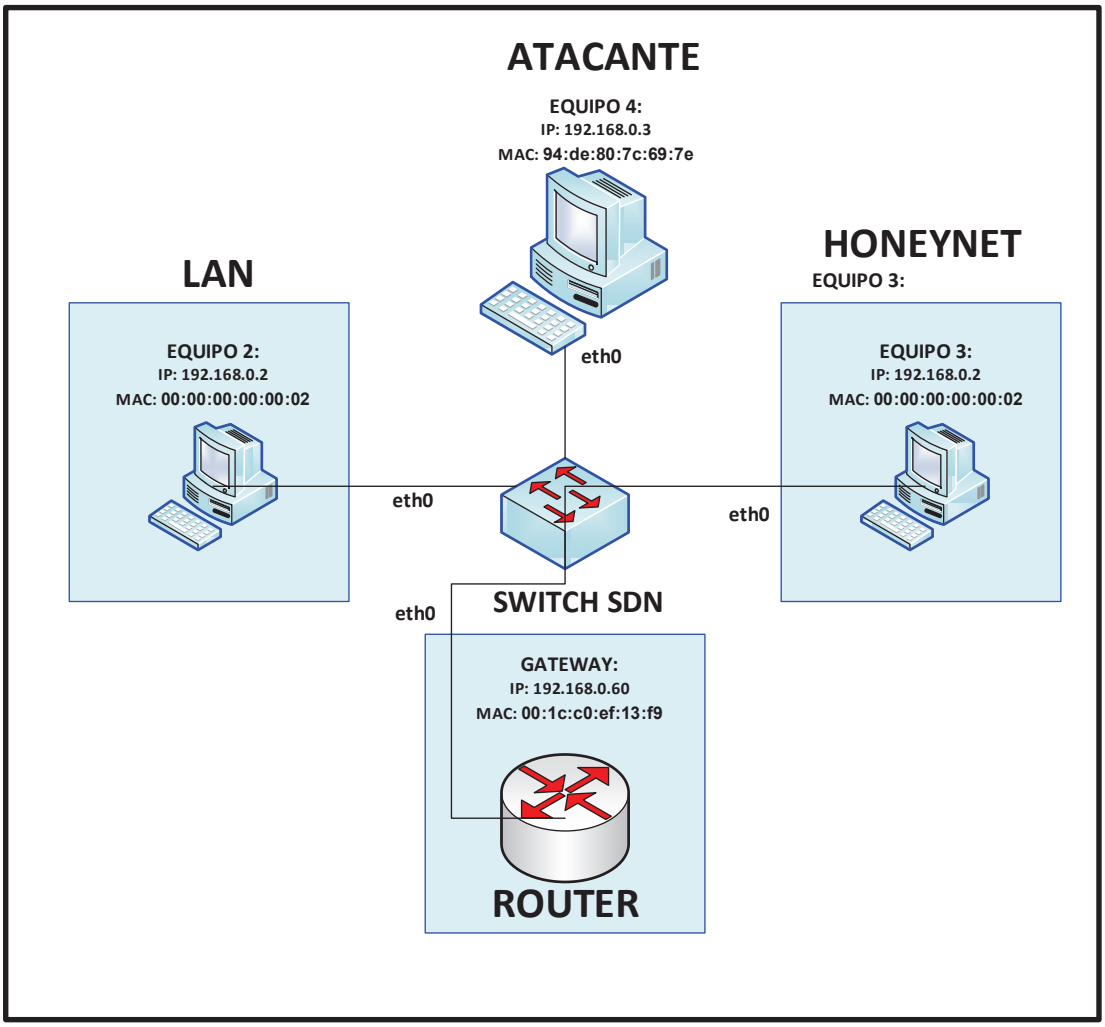

Figura 4.1 Diagrama utilizado para realizar las pruebas del ataque ARP_SPOOFING

Debido a la falla de diseño del protocolo ARP, que permite sin ninguna restricción actualizar las tablas ARP de los equipos cuando se recibe paquetes del tipo ARP_REPLY sin ninguna comprobación adicional, resulta la única opción efectiva para contrarrestar este ataque el uso de tablas ARP estáticas (configuradas

manualmente), sin embargo con el algoritmo implementado, se logra evadir el ataque casi en su totalidad.

En la Figura 4.2, muestra la captura de paquetes realizada en el equipo 2 con la herramienta Wireshark, se pueden observar 4 paquetes de datos, el primero es un paquete del tipo `ARP_REQUEST`, proveniente del atacante (dirección IP 192.168.0.3) y solicitando la dirección MAC del equipo cuya dirección IP es 192.168.0.60 (equipo suplantado).

El segundo paquete proviene del atacante, se trata de un paquete del tipo `ARP_REPLY`, y es un paquete para realizar el ataque, se aprecia que está dirigido al equipo cuyo dirección MAC es 00:00:00:00:00:02 (equipo 2, víctima del ataque), en el que se le informa que el equipo cuya dirección IP es 192.168.0.60, posee la dirección MAC 94:de:80:7c:69:7e (dirección MAC del atacante), es decir el mismo equipo que realizó la solicitud está contestando.

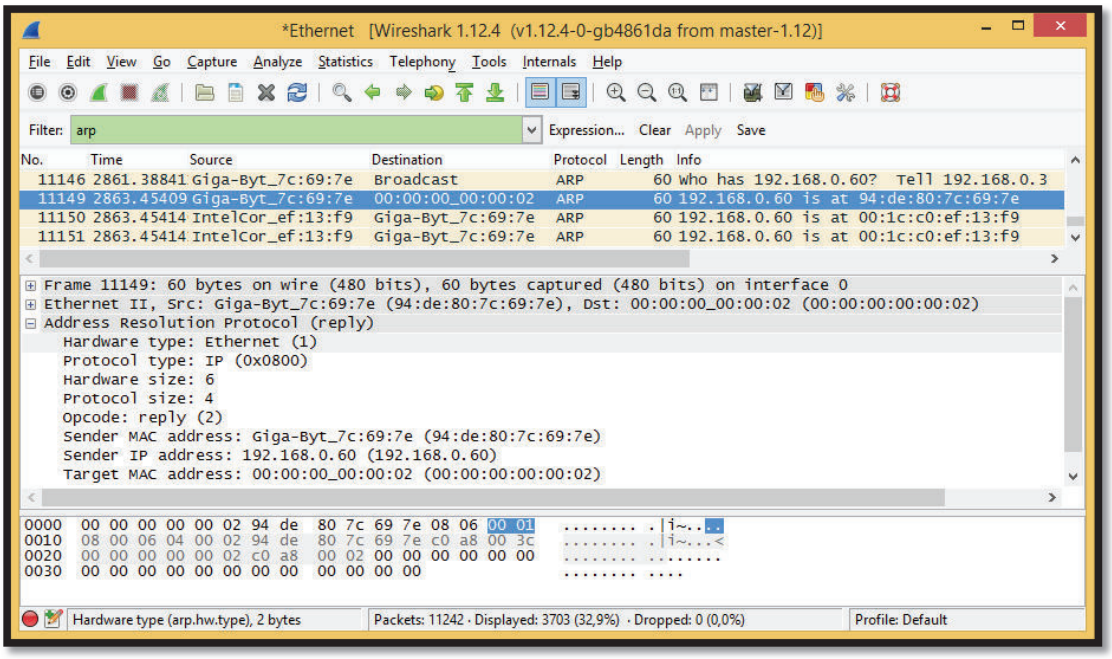

Figura 4.2 Captura de datos en la LAN con la herramienta Wireshark

Es claro que se trata de un atacante y el controlador procede a desviar todo el tráfico proveniente de este usuario hacia la *honeynet*, por lo tanto todos los demás

paquetes del tipo `ARP_REPLAY` recibidos posteriormente (en la gráfica se muestran 2) en este equipo de la LAN provenientes del equipo cuya dirección IP es 192.168.0.60 (router) indican que la dirección MAC asociada a la misma es 00:1c:c0:ef:13:f9, que efectivamente es la correcta. El tráfico capturado a continuación en la LAN no presenta ningún evento extraño.

Una vez que se ha identificado a un equipo como atacante, se procede a desviar el tráfico que provenga de ese usuario hacia la *honeynet* y en el equipo réplica de la *honeynet*, se empieza a capturar paquetes ARP como se muestra en la Figura 4.3.

Todos estos paquetes capturados en la *honeynet* (equipo 3) son paquetes enviados para realizar el ataque, pues provienen del equipo 4 (atacante) cuya dirección MAC es 94:de:80:7c:69:7e en los que se indica que el equipo con dirección IP 192.168.0.60 posee su misma dirección MAC. Dichos paquetes son dirigidos al equipo cuyo dirección MAC es 00:00:00:00:00:02 (equipo victima).

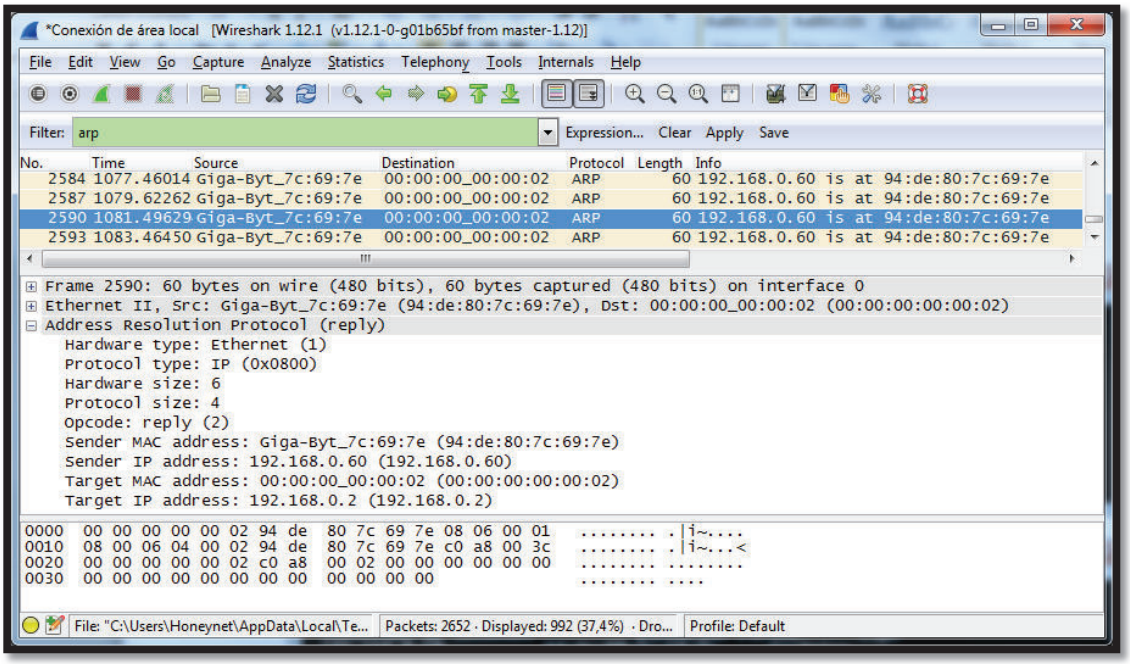

Figura 4.3 Captura de datos en el equipo réplica de la *honeynet*

La Figura 4.4 muestra la tabla ARP del equipo réplica de la *honeynet*, en la cual se aprecia que fue víctima del ataque, en esta se observa que efectivamente la dirección IP 192.168.0.60 se encuentra asociada a la dirección MAC del atacante.

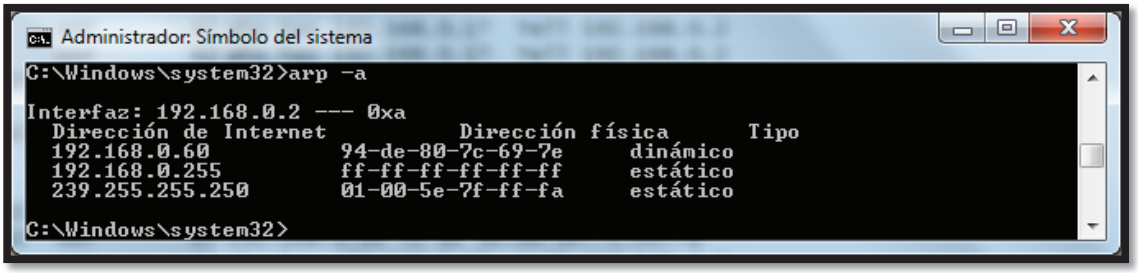

Figura 4.4 Tabla ARP del equipo réplica de la *honeynet*

Respecto a las estadísticas obtenidas en la *honeynet* durante la realización del ataque, como se observa en la Figura 4.5, de un total de 311 paquetes del tipo ARP capturados (color rojo), que representan el 29% del tráfico total capturado durante el ataque, 257 de estos son del tipo `ARP_REPLY` (color azul), y solamente los 54 paquetes ARP restantes, son del tipo `ARP_REQUEST` (color verde), lo que implica que del total de paquetes ARP recibidos, apenas el 17.36% fueron solicitudes ARP y el 82.64% corresponden a respuestas ARP.

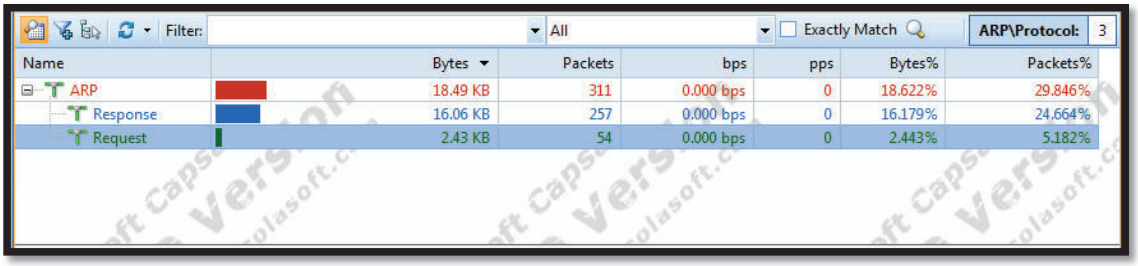

Figura 4.5 Estadísticas de paquetes ARP recibidos en el equipo víctima

Estos datos corroboran la falla de diseño de ARP, en la que un host puede recibir un número de paquetes `ARP_REPLY` mayor al número de paquetes `ARP_REQUEST` que ha generado y actualizar las tablas, sin determinar que se trata de un ataque.

4.10 ESTADÍSTICAS OBTENIDAS DEL ATAQUE IP SPOOFING

Las pruebas realizadas para determinar el correcto funcionamiento de la aplicación desarrollada ante el ataque `IP_SPOOFING`, se las efectuó utilizando la topología mostrada en la Figura 4.6, los equipos utilizados son los siguientes:

El equipo 2 posee la dirección IP 192.168.0.2 y la dirección MAC 00:00:00:00:00:02 (equipo víctima).

El equipo 3 es una réplica exacta del equipo 2 (honeynet).

El equipo 4 es el atacante, posee la dirección IP 192.168.0.3 y la dirección MAC 94:de:80:7c:69:7e.

El router (equipo cuya identidad será suplantada por el atacante) posee la dirección IP 192.168.0.1 y la dirección MAC 00:13:46:cf:08:84.

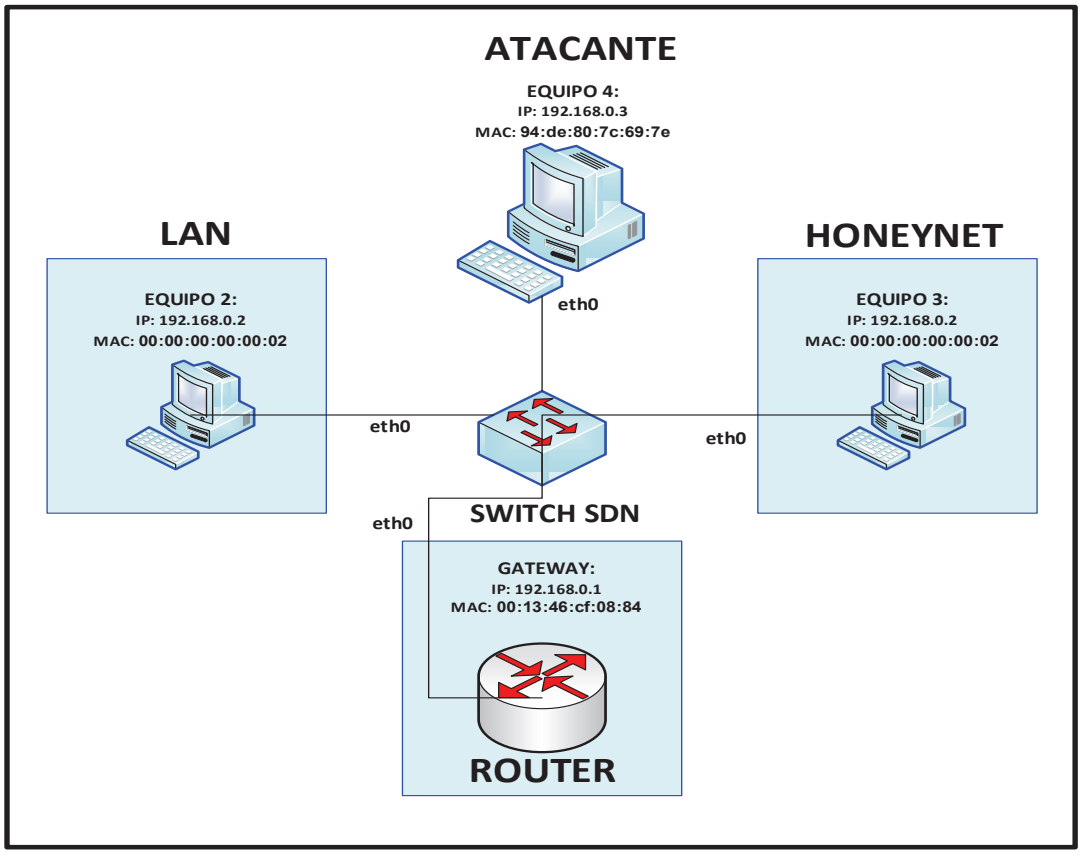

Figura 4.6 Topología utilizada para la realización de pruebas de funcionamiento para el ataque IP_SPOOFING

Tanto en la LAN como en la *honeynet* se tiene corriendo la aplicación Wireshark para realizar la captura de paquetes.

Las pruebas de suplantación de identidad `IP_SPOOFING` se realizaron utilizando el protocolo ICMP y el protocolo TCP.

A continuación se presentan los datos obtenidos al realizar el ataque utilizando el protocolo ICMP, el mismo que corre sobre el protocolo IP:

Durante el ataque no se observa ninguna variación en el tráfico que llega a la LAN, debido a que al detectar el ataque se lo desvía inmediatamente hacia la *honeynet*, por tal razón no se presenta ninguna captura de datos realizada en esta.

En la Figura 4.7 se puede observar el tráfico ICMP del ataque capturado con Wireshark en la *honeynet*, una vez que se lo ha desviado hacia la misma.

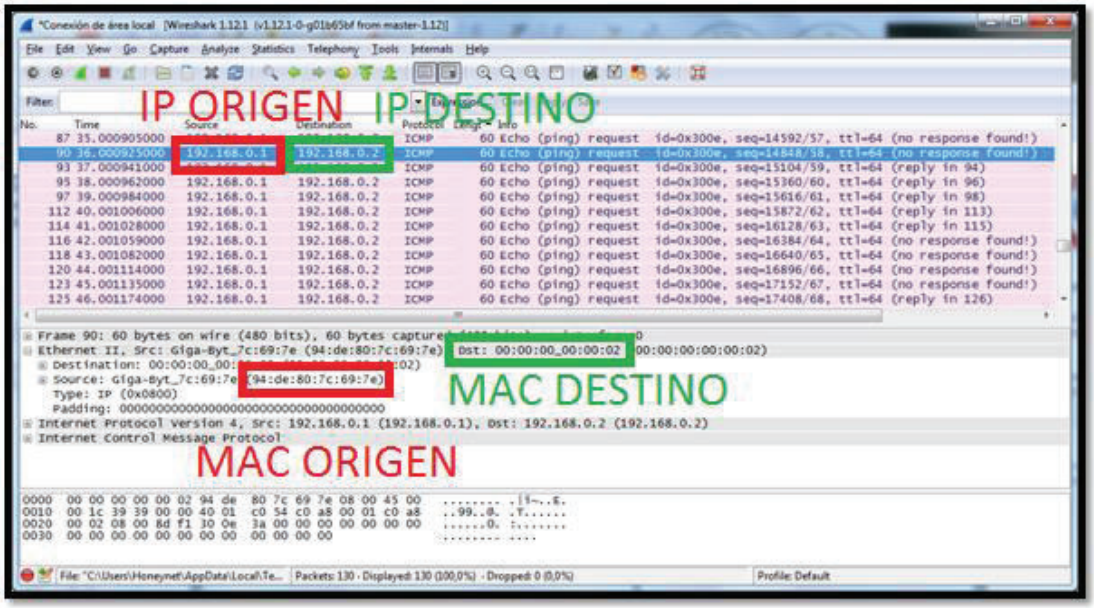

Figura 4.7 Captura de datos en el equipo víctima de IP_SPOOFING (ICMP_REQUEST)

Se observa que los paquetes capturados tienen como origen la dirección IP 192.168.0.1 (*gateway*) y la dirección MAC 94:de:80:7c:69:7e (dirección MAC del atacante), es decir los paquetes provienen del atacante pero con la dirección IP del *gateway*, esto se puede comprobar al observar la Figura 4.8, en la cual se muestra la tabla ARP del equipo 2 (víctima del engaño) en la que se observa que la dirección MAC asociada a la dirección IP 192.168.0.1 corresponde a la dirección MAC 00:13:46:cf:08:84, que es la dirección MAC verdadera, por tal motivo se determina que se trata de un paquete en el cual se está suplantando la identidad del *gateway*. También se ve que los paquetes suplantados se dirigen al equipo cuya dirección IP es 192.168.0.2 y que posee la dirección MAC 00:00:00:00:00:02.

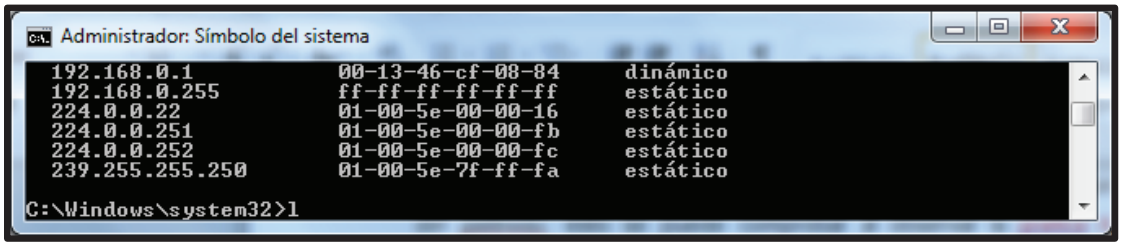

Figura 4.8 Tabla ARP del equipo 2

En la Figura 4.9 se puede observar que el equipo cuya dirección IP es la 192.168.0.2 (equipo atacado) al cual se enviaron los paquetes en los que se suplanto la identidad del *gateway*, procede a contestar, sin embargo, se ve claramente que éste envía las respuestas dirigidas a la dirección IP de la cual recibió los paquetes (IP suplantada del *gateway*), pero la dirección MAC de destino no corresponde a la dirección de la cual recibió los paquetes (dirección MAC del atacante), sino que se los envía a la dirección MAC asociada a la dirección IP que tenga en su tabla ARP, que se puede observar en la Figura 4.8

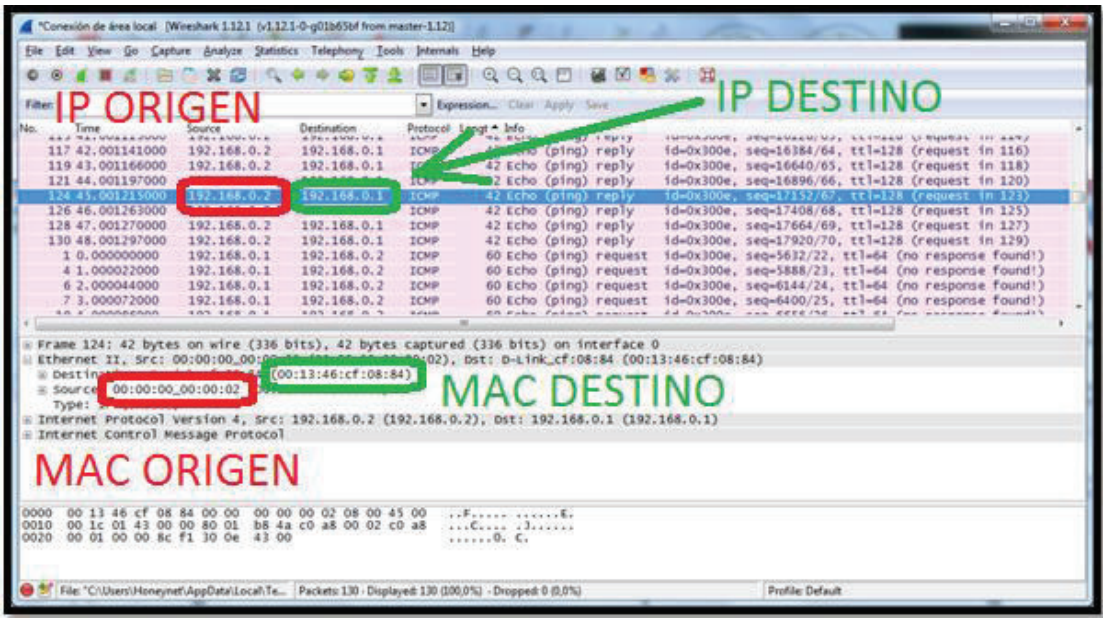

Figura 4.9 Captura de datos en el equipo víctima de `IP_SPOOFING` (`ICMP_REPLY`)

Esto se debe a que antes del ataque de tipo `IP_SPOOFING`, no se realizó un ataque de tipo `ARP_SPOOFING`, por lo tanto el equipo cuya dirección IP es la 192.168.0.2

tiene en su tabla MAC la verdadera dirección MAC del *gateway,* y al recibir los paquetes del tipo `ICMP_REQUEST`, este responde al verdadero equipo.

La segunda forma usada para probar el correcto funcionamiento del módulo para la detección de este ataque, se realizó utilizando el protocolo TCP. A continuación se muestran las capturas de datos que se obtuvieron al realizar las pruebas:

La Figura 4.10, muestra el trafico TCP capturado con Wireshark en la *honeynet*, una vez que se ha detectado el ataque y se lo ha desviado hacia la misma.

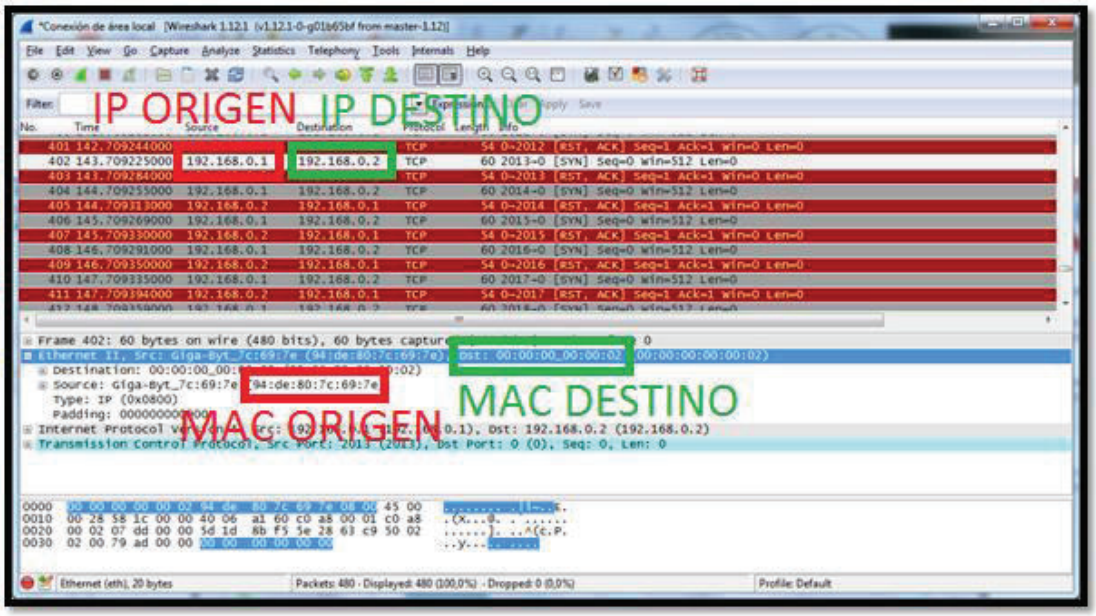

Figura 4.10 Captura de datos en el equipo víctima de `IP_SPOOFING` (`TCP_SYN`)

En esta se puede observar que se envían desde el atacante (dirección MAC origen 94:de:80:7c:69:7e) paquetes de intentos de conexión `TCP_SYN` en la que se suplanta de la misma manera que en el caso anterior la dirección IP del *gateway*, y se los dirige hacia el equipo 2.

Cuando el equipo 2 recibe los paquetes, envía un paquete TCP del tipo `TCP_RST`, para reiniciar la conexión, sin embargo como se observa en la Figura 4.11, dichos paquetes están dirigidos a una dirección MAC diferente a la dirección MAC de los paquetes `TCP_SYN` recibidos, los cuales fueron mostrados anteriormente.

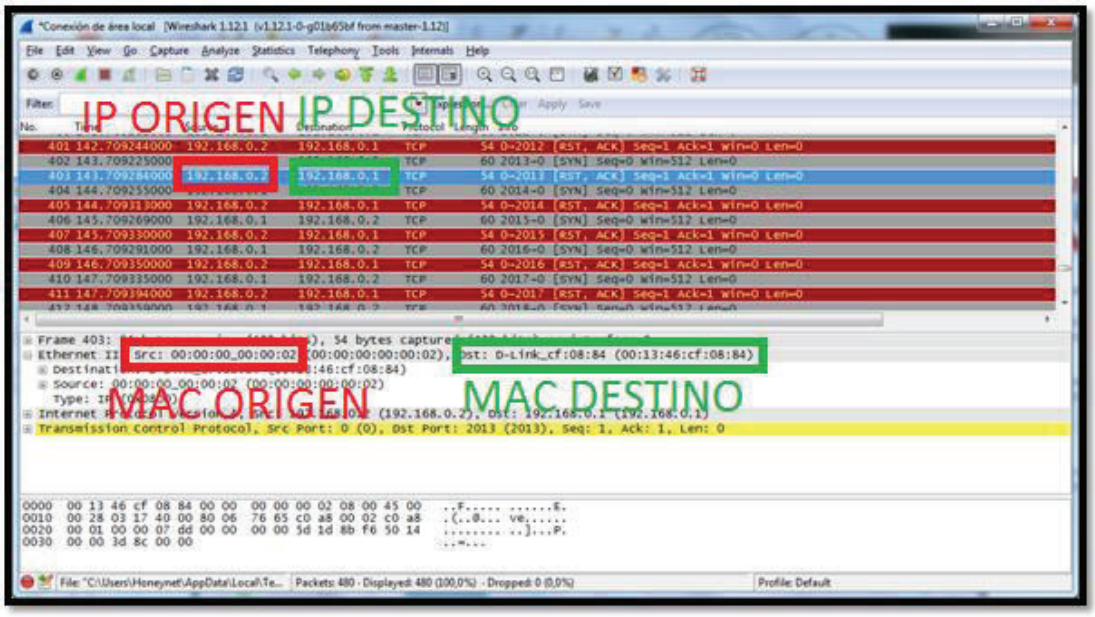

Figura 4.11 Captura de datos en el equipo víctima de `IP_SPOOFING` (RST, ACK)

La dirección MAC destino (mostrada en color verde), corresponde a la dirección MAC verdadera del *gateway.* Esto se puede comprobar si se observa en Figura *4.8* la dirección MAC asociada a la dirección IP del *gateway* (192.168.0.1).

Finalmente, en la Figura 4.12 se aprecia el tipo de tráfico que llega a la *honeynet*. Como se puede observar en la parte superior, todo el tráfico recibido en la *honeynet*, corresponde al protocolo IP, y en este se encuentra encapsulado TCP, esto se debe a que el único tráfico que se está dejando pasar a la honeynet, es el proveniente del ataque que se efectúa en ese momento.

En la parte inferior de la figura, se observa que la dirección IP 192.168.0.1 (dirección IP del *gateway*), se encuentra asociada a dos direcciones MAC diferentes, la del atacante y la del equipo propietario de la misma (*gateway*).

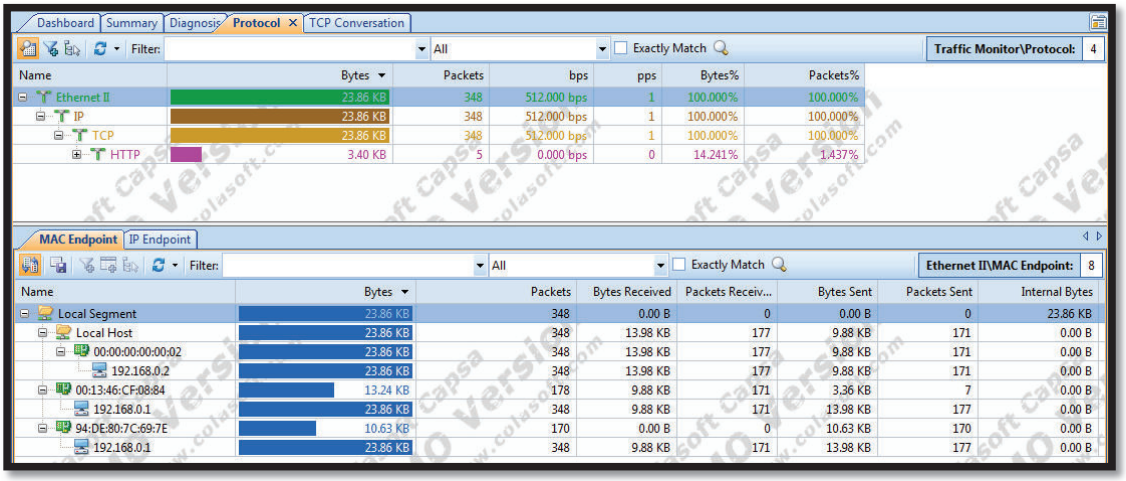

Figura 4.12 Captura de datos en el equipo víctima de `IP_SPOOFING` (TCP)

4.11 ESTADÍSTICAS OBTENIDAS DEL ATAQUE TCP SYN FLOOD

A continuación se presentan los datos obtenidos durante las pruebas de funcionamiento del módulo que permite detectar el ataque `TCP_SYN_FLOOD` cuando se encuentra en ejecución.

En la Figura 4.13, se puede apreciar la topología utilizada al realizar las pruebas de funcionamiento de este módulo, en la que se observa que los equipos utilizados fueron el equipo 2, 3 y 4.

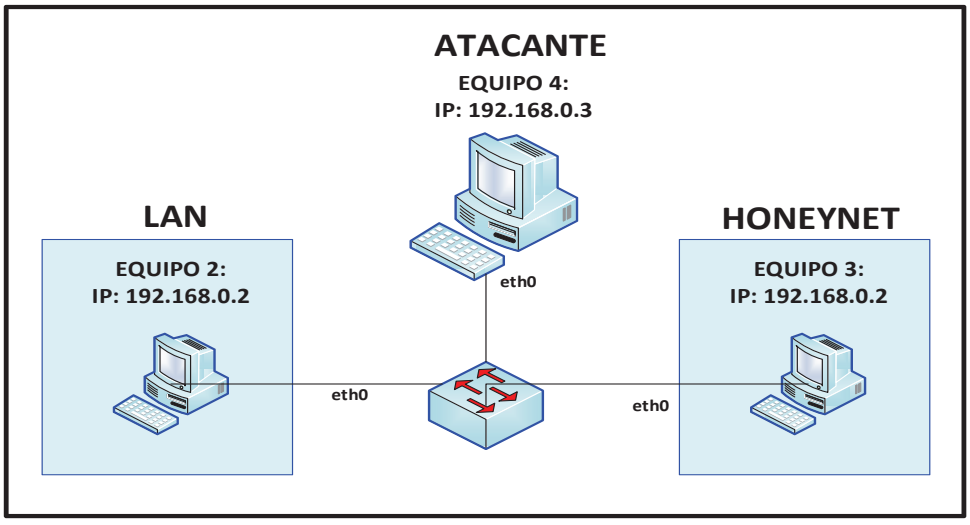

Figura 4.13 Topología utilizada para la realización de pruebas de funcionamiento del módulo `TCP_SYN_FLOOD`

El equipo 2 tiene configurada la dirección IP 192.168.0.2 y representa a la LAN, tiene instalado un servidor Web, el cual será se pretende atacar.

El equipo 3 es la réplica exacta del equipo 2 y representa a la honeynet.

El equipo 4 cuya dirección IP es 192.168.0.3, es el equipo desde donde se hace el ataque.

Tanto en la LAN como en la *honeynet*, se tiene corriendo la aplicación Wireshark que nos permite capturar el tráfico que llegue a cada uno de los equipos.

En la LAN, no se muestra ninguna variación respecto al tráfico de datos normal, esto debido a que al detectase el ataque, se procede a desviar el tráfico proveniente del equipo que ha sido determinado como atacante. Por tal razón no se ha creído conveniente presentar ninguna captura de datos.

En la *honeynet*, una vez que se ha detectado el ataque, se empieza a recibir información en el equipo réplica, en la Figura 4.14, se presenta la gráfica del tráfico debido a los paquetes del tipo `TCP_SYN` que se capturan una vez se inicia el ataque. En el eje horizontal se presenta una escala temporal, mientras que en el eje vertical, se presenta la cantidad de paquetes que se reciben; como se puede apreciar, la detección del ataque se realizó pasado las 20:07:43 y se registran intervalos de tiempo en los que se recibe hasta 165 paquetes de datos de intento de conexión (`TCP_SYN`).

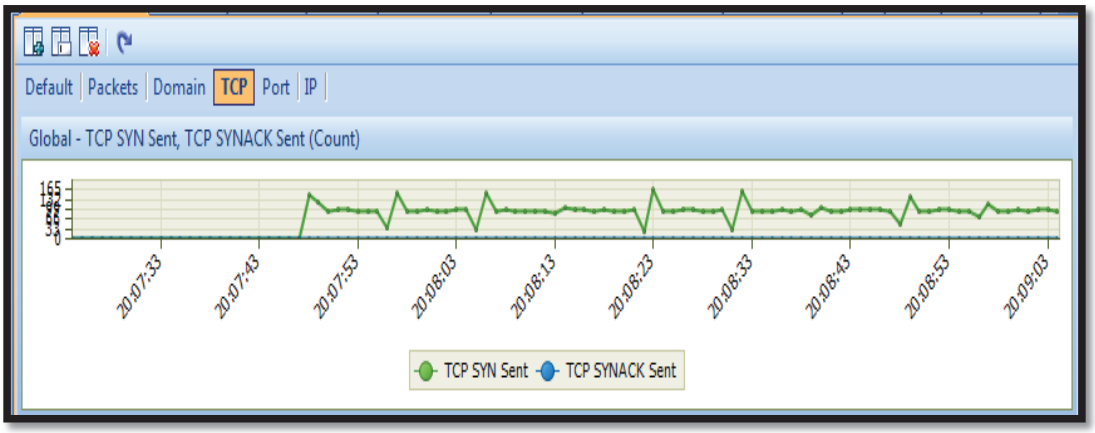

Figura 4.14 Captura de tráfico TCP en el equipo réplica de la *honeynet*

En la Figura 4.15, se presenta la captura de datos que se realizó con Wireshark en la *honeynet* durante el ataque, todos los paquetes que se reciben son del tipo `TCP_SYN` (primera vía de conexión), provienen del equipo cuya dirección IP es 192.168.0.3 (atacante) y se dirigen al equipo cuya dirección IP es 192.168.0.2 (víctima).

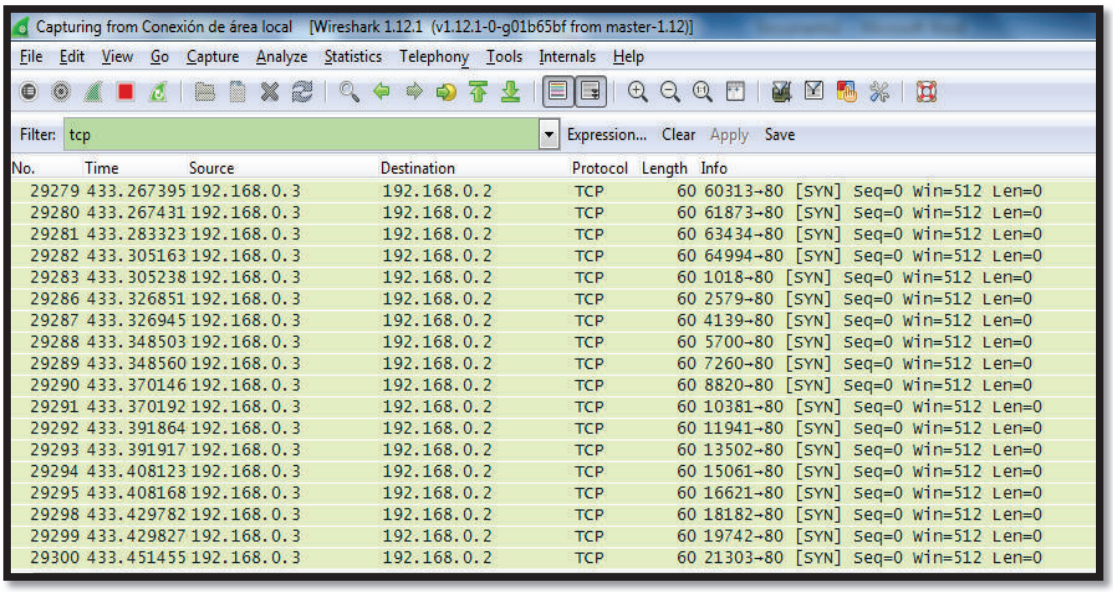

Figura 4.15 Captura de tráfico TCP durante el ataque

Resulta importante también analizar el tráfico capturado durante el ataque en función del tiempo, en la Figura 4.16 se observa la cantidad de tráfico que llega a la honeynet, en esta se puede apreciar que la actividad de la red aumenta considerablemente respecto a las condiciones normales de trabajo, esto provoca la sobrecarga del equipo por el procesamiento de solicitudes falsas se impide el procesamiento de conexiones legítimas.

La disminución que se observa en la gráfica en el intervalo de 460s a 480s se debe a que el equipo colapsa e intenta recuperarse y nuevamente procesar las peticiones que le continúan llegando.

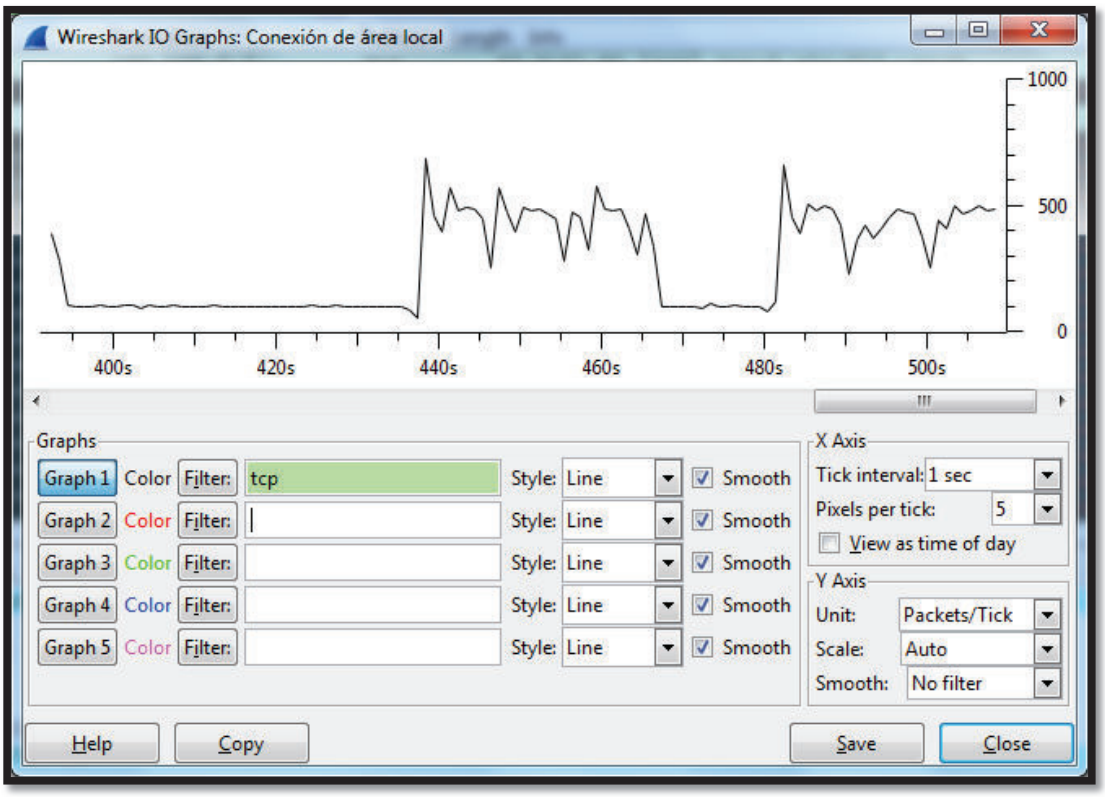

Figura 4.16 Variación del tráfico de red en la *honeynet*

Finalmente, se presenta las estadísticas obtenidas en la *honeynet* en cuanto a tipos de protocolos que se recibieron en la misma. En la Figura 4.17, se aprecia los diferentes tipos de paquetes recibidos durante el ataque en el equipo víctima; se ve que de un total de 8 094 paquetes Ethernet capturados, 7 445 encapsulan paquetes del tipo TCP, es decir el 91.98% del tráfico total que circula en la honeynet corresponde a tráfico TCP.

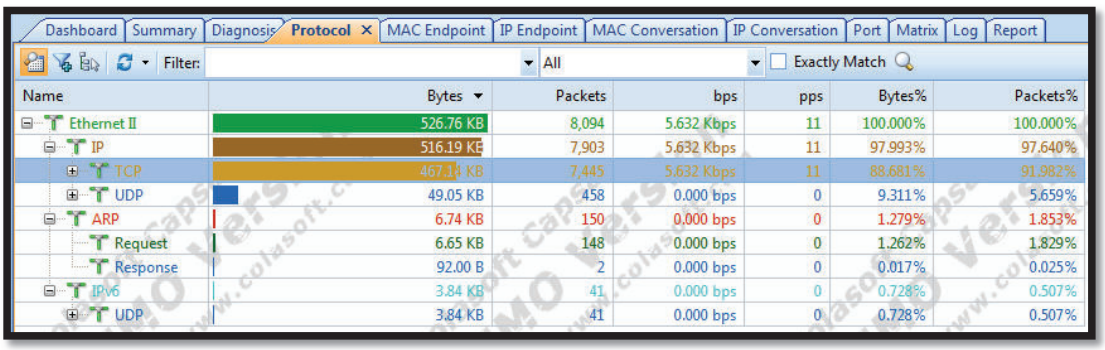

Figura 4.17 Estadísticas de los paquetes capturados en el equipo victima

De acuerdo a la captura de datos mostrada en la Figura 4.15, en la que se ve claramente que todos los paquetes que se capturan corresponden al tipo `TCP_SYN`, se determina que todo el tráfico TCP capturado en la honeynet es tráfico malicioso preveniente del atacante.

4.12 ESTADÍSTICAS OBTENIDAS DEL ATAQUE DNS_SPOOFING

Para la realización de pruebas de funcionamiento del módulo encargado de detectar este ataque, se utilizó la topología mostrada en la Figura 4.18.

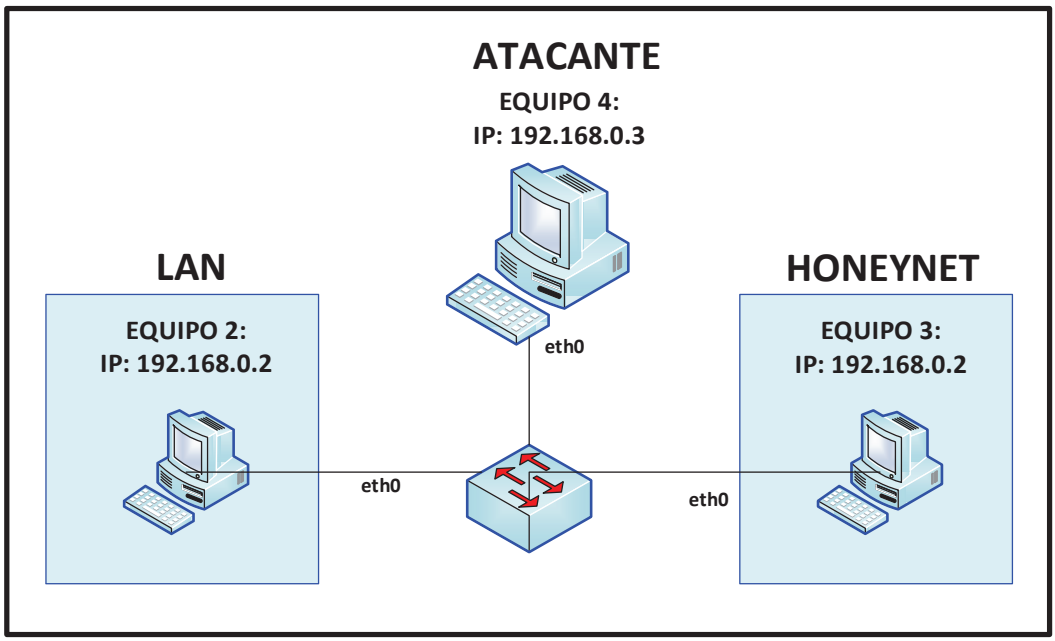

Figura 4.18 Topología utilizada para las pruebas de funcionamiento del módulo `DNS_SPOOFING`

Los equipos que se utilizaron fueron los siguientes:

El equipo 2 fue configurado con una dirección IP 192.168.0.2, representa a la LAN.

El equipo 3 es una réplica exacta del equipo 2 y representa la *honeynet.*

El equipo 4 cuya dirección IP es 192.168.0.3, representa al atacante.

Tanto en la LAN como en la honeynet, se tiene corriendo la aplicación que permite capturar el tráfico que circule por ellas. La variación del tráfico debido a este ataque es imperceptible en la LAN, sin embargo, en la Figura 4.19 se muestra la captura

de tráfico en la LAN con Wireshark. Como se puede observar, todos las solicitudes DNS generadas, se dirigen a la dirección IP 192.168.0.1 si se llegara a detectar que proviene de un equipo distinto al configurado como servidor DNS, entonces se desvían dichos paquetes de datos provenientes del atacante hacia la *honeynet*, en tal caso, se puede observar en la Figura 4.20, que se empieza capturar el tráfico DNS malicioso en la victima.

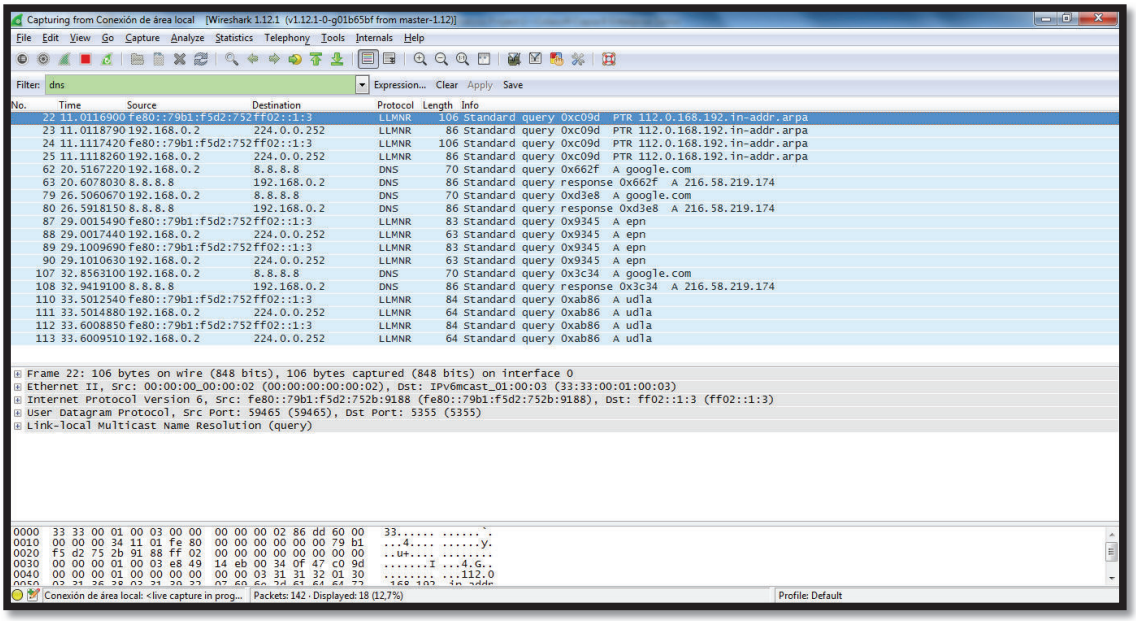

Figura 4.19 Peticiones DNS desde la LAN

Por lo tanto los picos producidos durante el intervalo de 0s a 40s mostrado en la Figura 4.20 , representan los paquetes correspondientes a respuestas DNS, provenientes del atacante

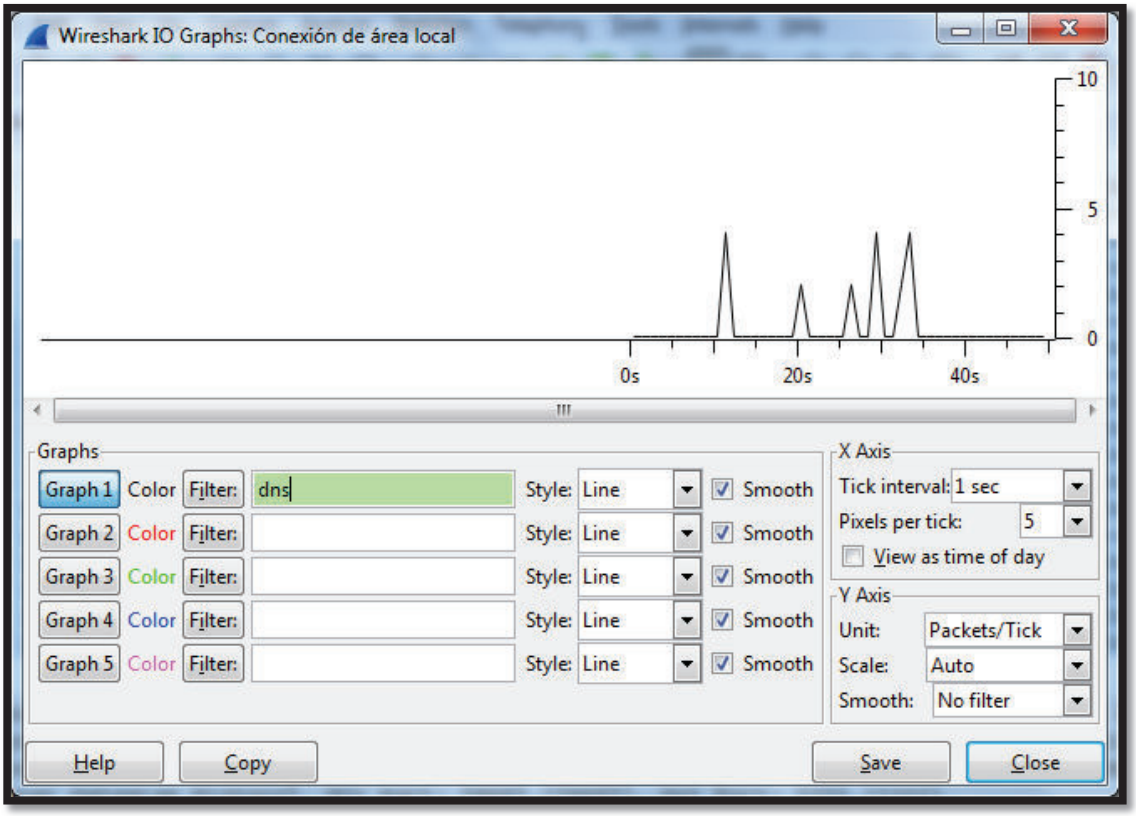

Figura 4.20 Trafico DNS capturado en el equipo víctima

4.13 ESTADÍSTICAS OBTENIDAS DEL ATAQUE SMURF

Para la realización de pruebas de funcionamiento del módulo encargado de detectar este ataque se utilizó la topología mostrada en la Figura 4.21.

Los equipos que se utilizaron fueron los siguientes:

El equipo 2 fue configurado con una dirección IP 192.168.0.2, representa a la LAN.

El equipo 3 es una réplica exacta del equipo 2 y representa la *honeynet.*

El equipo 4 cuya dirección IP es 192.168.0.3, representa al atacante.

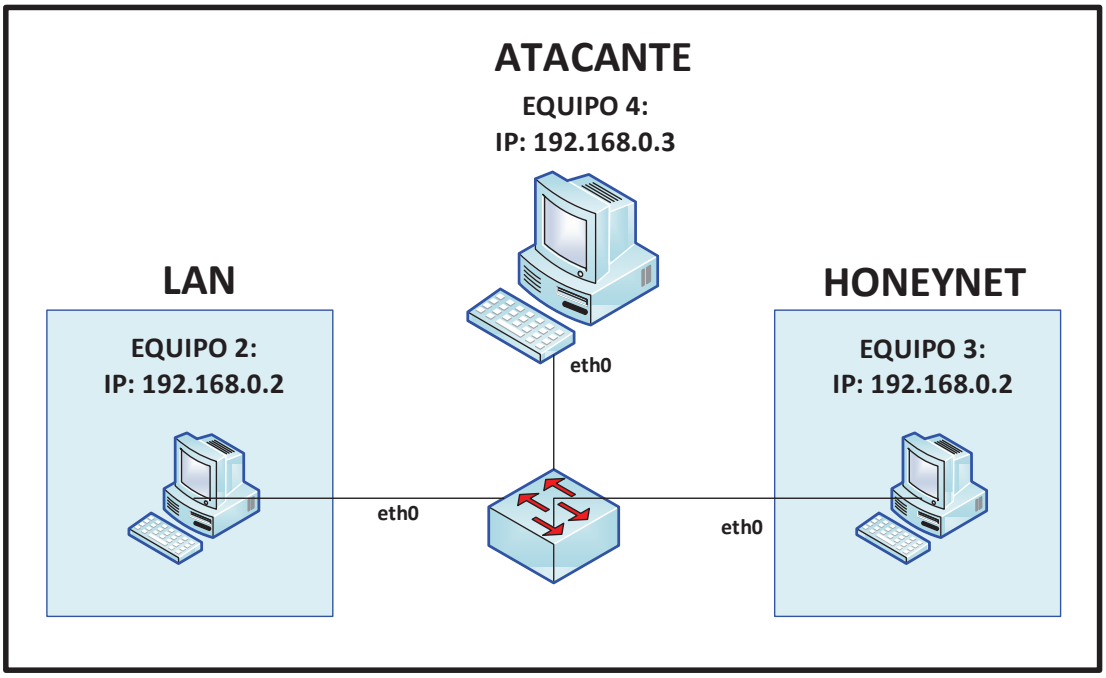

Figura 4.21 Topología utilizada para las pruebas de funcionamiento del módulo encargada de detectar el ataque SMURF

Tanto en la LAN como en la *honeynet*, se tiene a la aplicación Wireshark realizando captura de paquetes.

En la LAN no se ve ninguna variación en el tráfico, ni ningún comportamiento diferente al normal, esto se debe a que inmediatamente se detecta el ataque, se procede a desviar los paquetes hacia la *honeynet* por tal razón no se ha creído importante presentar información obtenida en la misma.

Sin embargo en la Figura 4.22, se puede observar la captura de datos que se realizó en la *honeynet* con la ayuda de Wireshark, en esta se puede apreciar que se recibe una gran cantidad de paquetes del tipo `ICMP_REQUEST` provenientes de la dirección IP de *broadcast,* y dirigidos a la dirección IP 192.168.0.2 (víctima).

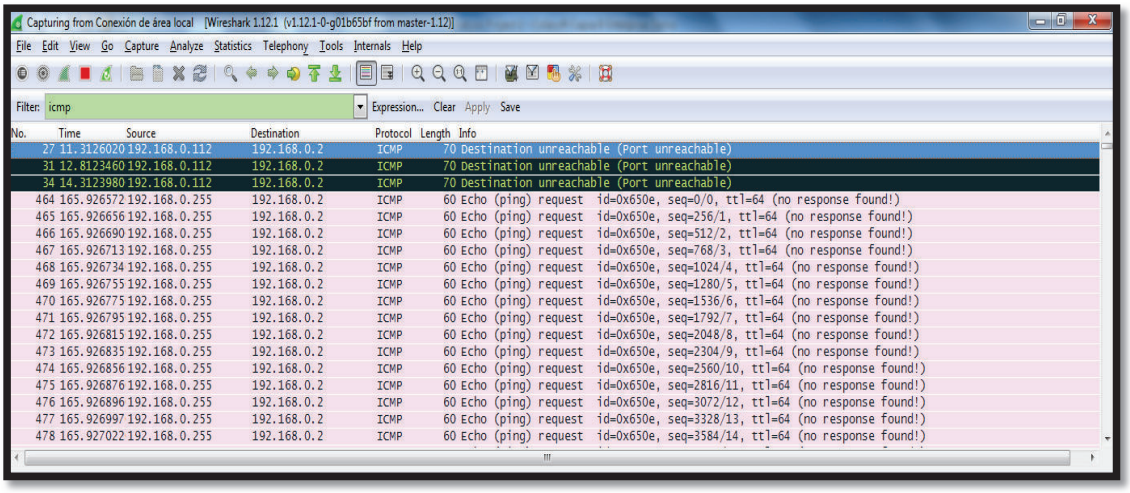

Figura 4.22 Captura de datos en el equipo víctima del ataque SMURF

La Figura 4.23, muestra la variación del tráfico ICMP con respecto a una escala de tiempo. En esta se puede apreciar dos pares de picos producidos entre los intervalos de tiempo, 160s a 180s y 250s a 260s. Cada par de picos producidos en los dos intervalo mencionados corresponden a como vario el tráfico durante los dos ataques consecutivos que se realizó.

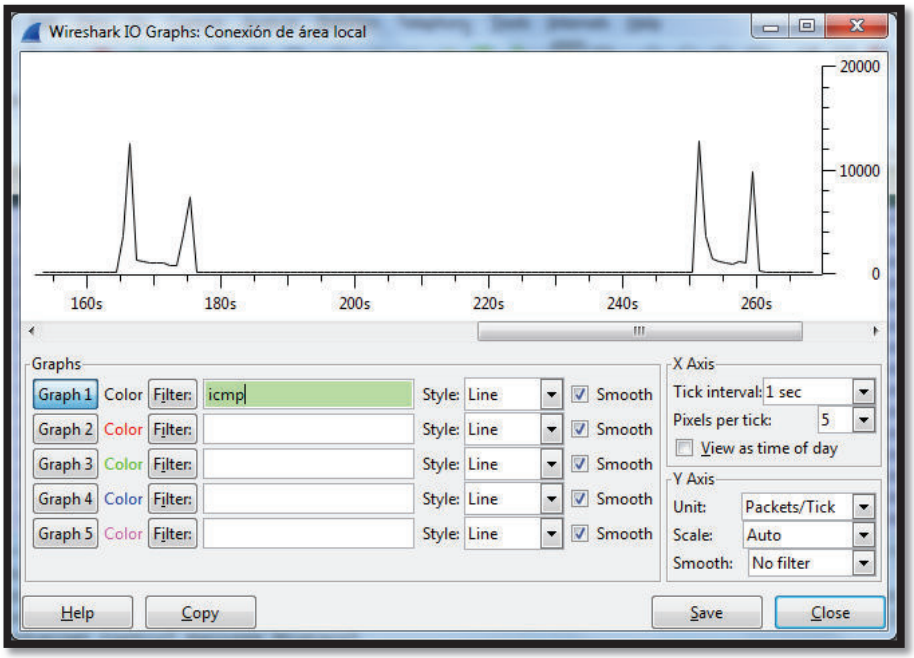

Figura 4.23 Variación del trafico ICMP durante el ataque SMURF

La sobrecarga producida en el equipo atacado es tan grande que lo deja inactivo por un lapso de tiempo, justamente el tiempo transcurrido entre 180s 250s en la misma gráfica, es por tal motivo que la actividad de captura de tráfico también se detiene hasta que el equipo vuelva a operar en condiciones normales.

Finalmente, en la Figura 4.24 se puede apreciar el tráfico capturado a partir de que inicia el ataque, corresponde en su mayoría a los paquetes del tipo `ICMP_REQUEST`, y que incorporan una dirección IP origen de *broadcast*.

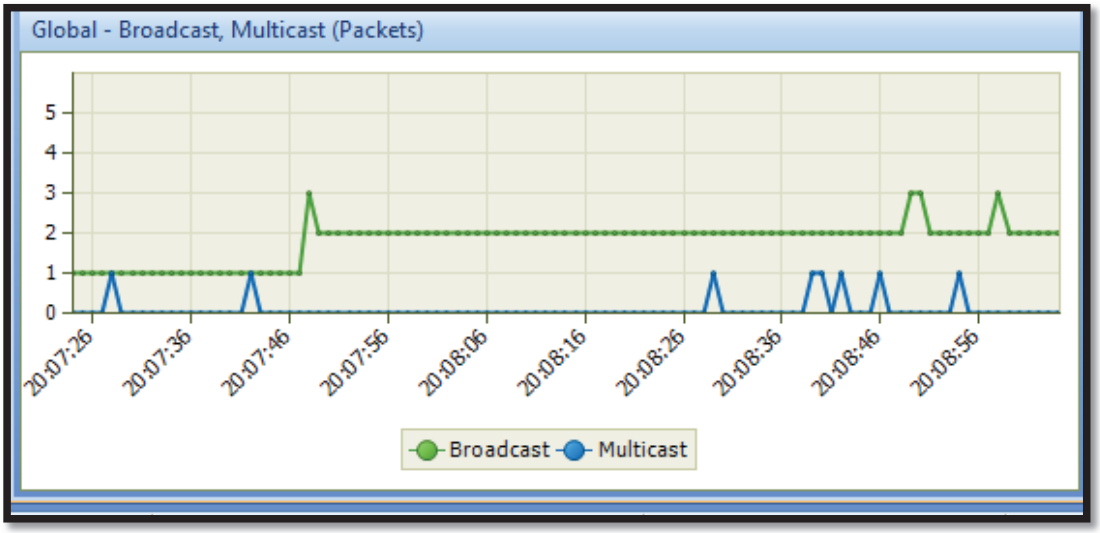

Figura 4.24 Variación del tráfico durante el ataque SMURF

4.14 ESTADÍSTICAS OBTENIDAS DEL ATAQUE THC SSL DoS

Para la realización de pruebas de funcionamiento del módulo encargado de detectar este ataque se utilizó la topología mostrada en la Figura 4.25.

Los equipos que se utilizaron fueron los siguientes:

El equipo 2 fue configurado con una dirección IP 192.168.0.9, representa a la LAN, en este equipo se encuentra instalado un servidor Web que trabajo en conjunto con el protocolo SSL.

El equipo 3 es una réplica exacta del equipo 2 y representa la *honeynet.*

El equipo 4 cuya dirección IP es 192.168.0.3, representa al atacante.

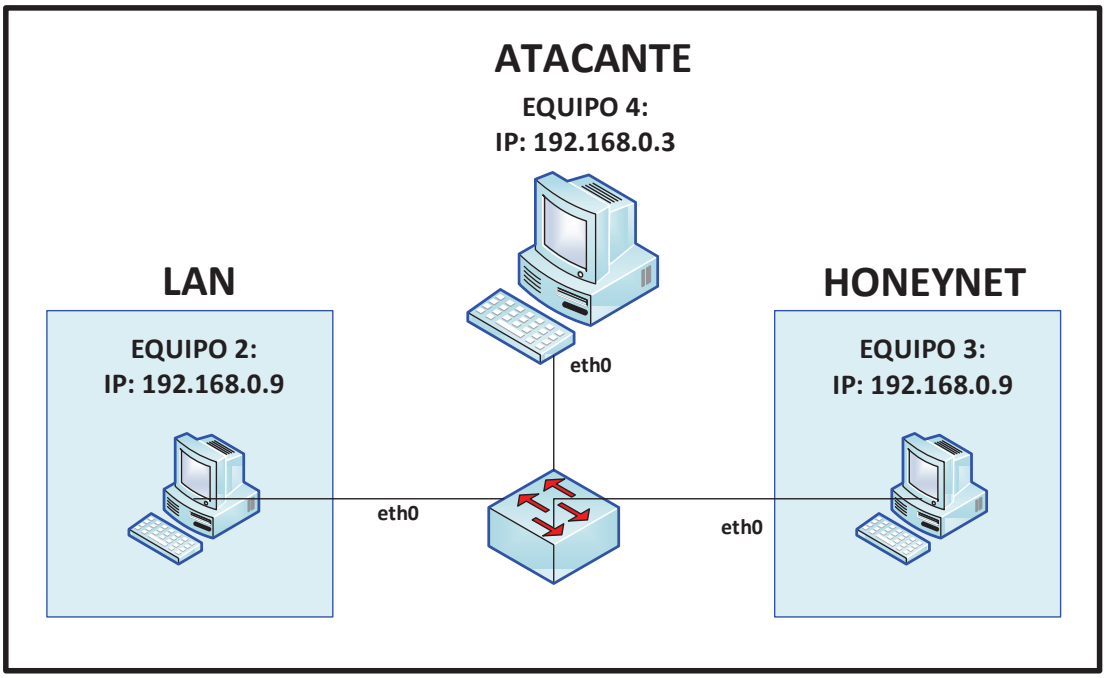

Figura 4.25 Topología utilizada para realizar las pruebas de funcionamiento del módulo en cargado de detectar el ataque THC SSL DoS

Tanto en la LAN como en la *honeynet*, se encuentra corriendo las aplicaciones que permiten capturar datos, como es Wireshark y Colasoft Capsa.

En la LAN, no se ha detectado ningún tipo de variación respecto a su comportamiento normal de la red, esto se debe a que inmediatamente se detecta el ataque se desvía los paquetes hacia la *honeynet*. Por tal razón no se ha incorporado estadísticas de la misma.

En la *honeynet*, inmediatamente se desvía el ataque, se empieza a tener actividad en el equipo víctima como se muestra en la Figura 4.26, en la que puede apreciar en las dos graficas superiores, el porcentaje de utilización y el tráfico total en bytes que se recibe en el quipo, mientras que en la gráfica inferior de la izquierda se ve que el tráfico de datos se efectúa en su mayoría entre el equipo cuya dirección IP es 192.168.0.9 (victima) y el quipo cuya dirección IP es 192.168.0.3 (atacante), finalmente en la gráfica inferior derecha se ve que el tráfico de datos corresponde al protocolo HTTPS.

Figura 4.26 Tráfico de datos en el equipo victima

En la Figura 4.27, se puede apreciar que del total de datos que se reciben encapsulados en TCP, el 60,34% de estos corresponden al protocolo HTTPS (color rojo), esto se debe a que a nivel de TCP también se intercambia tráfico de *handshake* para establecer la conexión y una vez establecida se empieza a enviar el tráfico HTTPS.

En la parte inferior de la misma gráfica, se ve que el intercambio de datos se realiza entre los equipos cuyas direcciones IP son 192.168.0.9 (victima) y 192.168.0.3 (atacante).

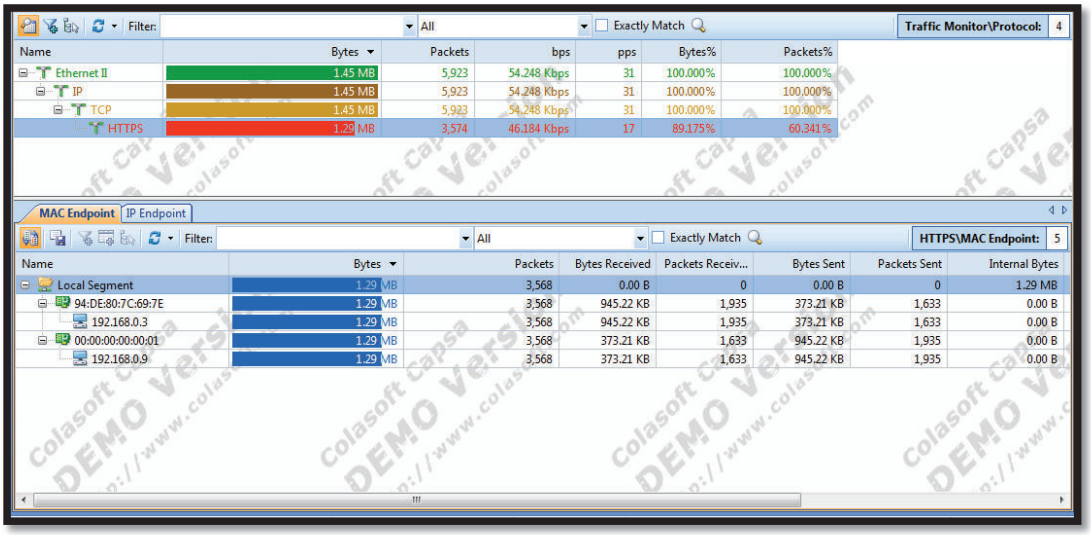

Figura 4.27 Captura de tráfico de datos en equipo victima

La Figura 4.28 muestra la depuración del módulo correspondiente, cuando se está enviando los paquetes al puerto 2 en el que se encuentra conectada la *honeynet*.

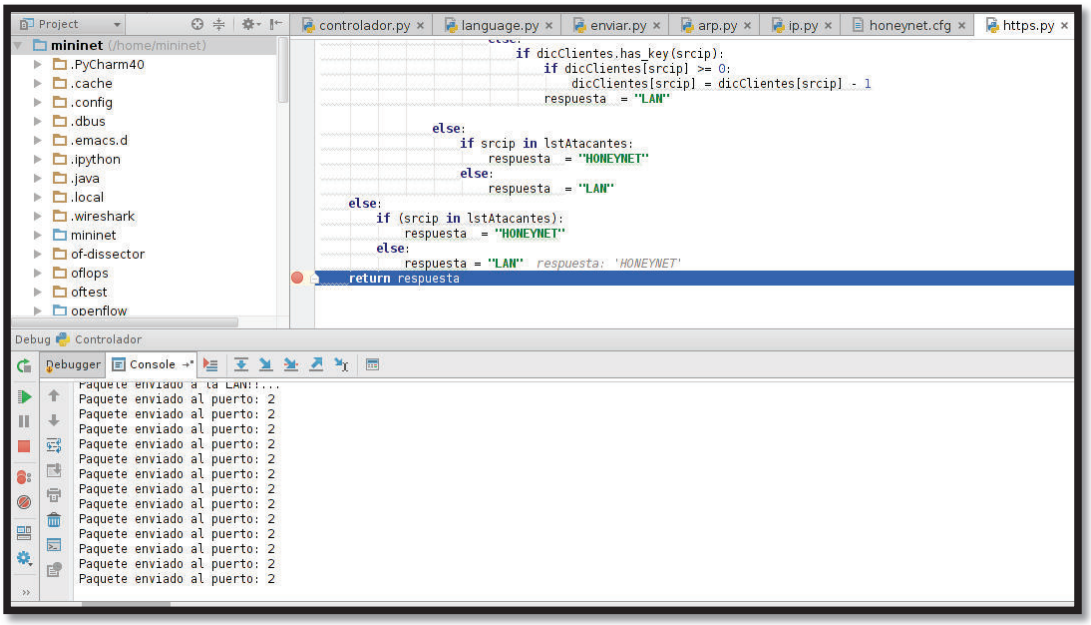

Figura 4.28 Depuración del módulo `https.py`

Finalmente, en la Figura 4.29 se muestra la variación del tráfico en la *honeynet* en función del tiempo. Como se puede observar, el ataque se efectuó durante el intervalo de tiempo comprendido entre 460s y 520s, en el que se ve un incremento considerable del tráfico que circula en el equipo victima respecto al comportamiento inicial.

También se puede ver claramente una leve variación del tráfico luego de que cesa el ataque, esto es debido a que el equipo atacado envía paquetes de reintento de conexión.

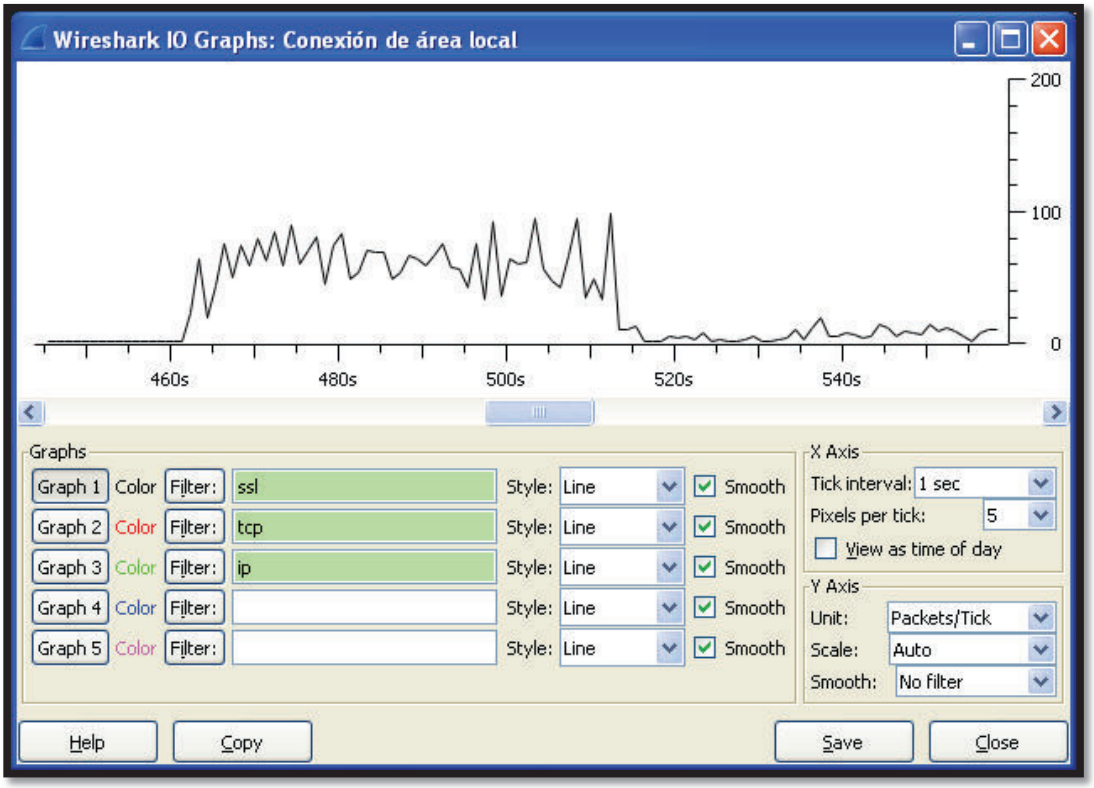

Figura 4.29 Variación de la cantidad de tráfico en el equipo víctima durante el ataque

CAPÍTULO V

5 CONCLUSIONES Y RECOMENDACIONES

Una vez se ha presentado toda la información obtenida a partir del prototipo desarrollado, se muestra a continuación las conclusiones y recomendaciones que han sido formuladas por los autores del prototipo:

5.1 CONCLUSIONES

- Al finalizar el presente proyecto se ha obtenido el prototipo de *honeynet* que utilizando Redes Definidas por Software (SDN), permite detectar seis diferentes tipos de ataques informáticos, entre los que se incluyen tres ataques del tipo denegación de servicio, y tres ataques de suplantación de identidad o *spoofing*, dicho prototipo ha sido el objetivo principal de este proyecto y actualmente se encuentra desarrollado, probado y funcionando adecuadamente, en las topologías de red que se definieron para cada uno de los ataques.
- Al utilizar las redes definidas por software el administrador de la red tiene el control completo de la red, los paquetes son tratados como él crea conveniente, lo que no sucede con las redes tradicionales, en las que el administrador tiene que acoplarse a los protocolos y tecnologías que en muchos de los casos son propietarios, lo que provoca no tener un conocimiento a fondo de cómo funciona dicho protocolo o no se pueda dar el tratamiento adecuado al tráfico, lo que a la larga puede provocar problemas como en la compatibilidad. Sin embargo la utilización de SDN presenta ventajas y desventajas como cualquier otra tecnología, por ejemplo resulta favorable el tener todo el control que se menciona si se crean las reglas de forma adecuada, pero si no se crean bien las reglas puede fallar toda la red, siendo esta ventaja también la mayor desventaja.
- El controlador utilizado para el desarrollo del prototipo llamado Pyretic, proporciona todas las herramientas necesarias para la implementación de la aplicación, dicho controlador ha sido modificado periódicamente y durante el transcurso de tiempo que tomo el desarrollo del proyecto, ha sufrido un sin

número de modificaciones que le han permitido incorporar nuevas funcionalidades y facilidades para la implementación de ciertas características, por lo tanto se concluye que el controlador seleccionado ha sido el adecuando, y no solo por sus herramientas sino también por el soporte proporcionado por los desarrolladores del mismo. Ha sido mucha la ayuda brindada por el grupo de trabajo que desarrolla el controlador, especialmente la proporcionada por el director del proyecto Dr. Joshua Reich de la universidad de Princeton, quien ha realizado ciertas observaciones y recomendaciones sobre el uso y funcionamiento del controlador.

- La implementación de cada uno de los módulos de detección de ataques informáticos requirió del análisis de los diferentes protocolos que trabajan en las distintas capas según el modelo de referencia ISO/OSI, dicho análisis, incorpora el estudio de los campos de las cabeceras y colas de los paquetes, los diferentes tipos de paquetes, los distintos escenarios de funcionamiento, es por tal motivo que en cada uno de ellos se ha requerido implementar un diferente algoritmo de trabajo, sin embargo la aplicación desarrollada permite la detección de los seis diferentes ataques informáticos en situaciones particulares que abarcan casi la totalidad de los caso, debe mencionarse por supuesto que existen como es en el caso de ARP, ciertas fallas de diseño del protocolo que no permiten la detección del ataque como tal, y ante esta falla, se ha conseguido minimizar en gran medida el efecto del ataque, logrando desviarse el trafico malicioso en alrededor de un 90% de los casos. Esta falla se produce debido a que los equipos pueden cambiar constantemente de dirección lógica(dirección IP), por lo que la dirección IP asociada a una dirección MAC puede cambiar de forma dinámica, esto permite que un ataque del tipo `ARP_SPOOFING` no pueda ser detectado, pues es difícil determinar si se trata de un ataque o simplemente de una nueva asignación de dirección IP a un determinado equipo, ya sea que se trate de una asignación dinámica (mediante un servidor de DHCP) o mediante una asignación estática (de forma manual).
- La implementación del ataque `ARP_SPOOFING`, ha requerido que se realice una revisión previa del protocolo ARP, principalmente para determinar su forma de funcionamiento; se ha logrado determinar que efectivamente la

única manera de contrarrestar este tipo de ataque es mediante la utilización de configuraciones estáticas de tablas ARP en los equipos, es decir asignadas de forma manual y permanente en cada equipo, esto permite que las tablas ARP de los diferentes host, permanezcan fijas y no se sean modificadas por el protocolo ARP, permitiendo una comunicación fiable entre los mismos, sin embargo este procedimiento, podría resultar tedioso y en un gasto de esfuerzo y tiempo considerable, especialmente cuando se esté trabajando en grandes redes de datos en las que participan miles de usuarios, en cuyo caso ante la necesidad de añadir o modificar la dirección de un usuario, se debería modificar las entradas estáticas de las tablas ARP en cada uno de los host que vayan a poder comunicarse con dicho host, por lo tanto resulta mucho más factible, rápido y útil la utilización de la solución presentada en este trabajo. Es también importante mencionar que este tipo de ataque informático únicamente funciona dentro de una misma subred de datos, es decir no puede ser implementado desde afuera de la LAN, pues este protocolo no es enrutable, por lo tanto el ataque debe realizarse desde un equipo que se encuentre en la misma subred.

- Se concluye que la implementación del módulo que permite detectar el ataque de `IP_SPOOFING`, puede trabajar en dos diferentes ambientes, esto es, cuando encapsula paquetes del tipo TCP y cuando encapsula paquetes del tipo ICMP, esto debido a que los diferentes casos en los que se puede realizar este tipo de ataque son innumerables, se decidió utilizar los dos casos más comunes para este fin.
- El modulo que permite determinar el ataque `TCP_SYN_FLOOD`, es un módulo muy dinámico, pues permite desviar paquetes que provengan desde un origen que haya sido considerado como atacante, siempre y cuando se esté realizando un ataque, sin embargo si dicho host actúa de forma normal (sin efectuar ningún tipo de ataque), entonces se lo vuelve a tratar como un cliente legítimo, de esta manera se evita la restricción del acceso a clientes que posiblemente pudieron sufrir de un secuestro de identidad momentáneo, pero que efectivamente se trata de un cliente legítimo.
- El presente proyecto es una muestra significativa de la preparación académica de los estudiantes de la Escuela Politécnica Nacional, que

permitió desarrollar un prototipo de *honeynet* utilizando redes definidas por software en la que se destaca la preparación de los estudiantes para enfrentarse ante nuevas tecnologías una vez que en su preparación académica de pregrado, han sido fortificados los conocimientos básicos de ingeniería y que son los que le permiten entender y acoplarse a nuevas tecnologías y protocolos de red, como es el caso del nuevo protocolo utilizado que es OpenFlow, dicha preparación también les permite acoplarse o entender fácilmente nuevos lenguajes de programación como fue en este caso, que los estudiantes desarrollaron su aplicación en el lenguaje de programación Python y finalmente también vale la pena mencionar la gran capacidad que cuentan los estudiantes para solucionar problemas.

5.2 RECOMENDACIONES

- Se recomienda hacer una investigación a fondo sobre el uso del controlador Pyretic, y determinar si es posible actualmente, realizar manejo de los paquetes en forma de múltiples procesos, mediante el manejo de hilos de ejecución diferentes por cada uno de los paquetes que se reciban en el switch SDN, esto podría ayudar a minimizar de forma significativa los retardos obtenidos en la aplicación desarrollada.
- El uso de un único equipo en el que se ejecute el controlador, puede ser un gran centro de falla que podría dejar inoperante a toda la red, es por tal motivo que se recomienda el uso de varios equipos en los que se corra el controlador de forma distribuida, esto para evitar que en caso de que ocurriera una falla en uno de los equipos la red no se quede inoperativa, y valiéndose de su naturaleza distribuida permita continuar con un funcionamiento tal vez un tanto limitado pero en un estado operable a la red.
- Se recomienda realizar pruebas de funcionamiento en entornos reales, pudiendo tratarse de pequeñas redes de datos, como es el caso de laboratorios de informática o redes de investigación pequeñas, esto podría ayudar a genera resultados novedosos del comportamiento de la red, de la forma de atacar de los intrusos y del nivel de protección que se le puede dar a la misma utilizando los algoritmos desarrollados en el presente proyecto.

- La utilización de SDN implica tener un amplio conocimiento de las redes de datos y de los diferentes protocolos, es importante utilizar herramientas que permitan analizarlos como es el caso de Wireshark por lo que se recomienda analizar previamente cada protocolo y todo lo que conlleva su funcionamiento, como es el caso de ARP, del cual todos los protocolos dependen para que los equipos se conozcan, y se pueda establecer las comunicaciones en capas superiores dentro de una mimo segmento de red.
- Se recomienda configurar una sola instancia de OpenFlow, en un mismo switch si se llegara a utilizar el modelo de switch que se utilizó en este proyecto: Se ha visto que muchas veces y debido a que la tecnología aún está en desarrollo, eventualmente se producen fallos de red, como es el caso de paquetes perdidos, en los que son entregados al controlador y este los devuelve al switch pero no se logran inyectar en la red, sin embargo en el controlador se puede tener registro de la cantidad de paquetes procesados y destinados a un equipo en particular y al mismo tiempo tener un *sniffer* en el equipo capturando los paquetes que le envía en switch y resulta que no corresponde al mismo número, de tal manera que se comprueba que existe perdida de paquetes.
- Se recomienda que el equipo en el que se corre el controlador sea un equipo con muy buenas características porque en este es donde se realiza la mayor cantidad de procesamiento y por tanto, al tener un tráfico bastante grande, este puede colapsar y dejar a toda la red sin conectividad, adicionalmente, se recomienda instalar el sistema operativo en el que se va a correr el controlador en una partición del disco duro totalmente independiente, esto debido a que el uso de máquinas virtuales limita en cierta medida la capacidad que se tenga en el equipo físico.

REFERENCIAS BIBLIOGRÁFICAS

[1] «Software Defined Network,» 01 05 2013. [En línea]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition. [Último acceso: 18 10 2014].

[2] «Source Forge,» 25 07 2003. [En línea]. Available: http://his.sourceforge.net/honeynet/papers/honeynet/. [Último acceso: 06 08 2014].

[3] «Search Security,» [En línea]. Available: http://searchsecurity.techtarget.com/definition/honeynet. [Último acceso: 07 08 2014].

[4] L. Spitzner, «Tracking Hackers,» de *Honeypots*, Addison Wesley, 2002.

[5] R. Siles, «redIRIS,» [En línea]. Available: https://www.rediris.es/cert/doc/reuniones/fs2007/archivo/Honeynets_RaulSiles_VForoSeguridadRedIRIS_Abril2007.pdf. [Último acceso: 13 08 2014].

[6] «Source Forge,» 12 05 2004. [En línea]. Available: http://his.sourceforge.net/honeynet/tools/. [Último acceso: 14 08 2014].

[7] «ebtables,» [En línea]. Available: http://ebtables.sourceforge.net/. [Último acceso: 21 08 2014].

[8] «Honeysnap,» [En línea]. Available: https://projects.honeynet.org/honeysnap/. [Último acceso: 07 12 2014].

[9] «Source Force,» [En línea]. Available: http://sourceforge.net/projects/ettercap/. [Último acceso: 06 08 2014].

[10] Irongeek, «http://www.irongeek.com/i.php?page=backtrack-3-man/ettercap,» [En línea]. Available:

http://www.irongeek.com/i.php?page=backtrack-3-man/ettercap. [Último acceso: 02 09 2014].

[11] P. Carrasco, «manual hping,» [En línea]. Available: http://www.pedrocarrasco.org/manual-practico-de-hping/. [Último acceso: 02 09 2014].

[12] RadarHack, «Tutorial Hping2,» [En línea]. Available: http://www.radarhack.com/tutorial/hping2.pdf. [Último acceso: 02 09 2014].

[13] thc.org, «THC-SSL-DOS,» [En línea]. Available: https://www.thc.org/thc-ssl-dos/. [Último acceso: 02 09 2014].

[14] searchsecurity, «Syn-Flood,» [En línea]. Available: http://searchsecurity.techtarget.com/definition/SYN-flooding. [Último acceso: 02 09 2014].

[15] H3C, «H3C,» [En línea]. Available: http://www.h3c.com/portal/Products___Solutions/Technology/Security_and_VPN/Technology_White_Paper/200812/624110_57_0.htm.

[16] J. Blasco, «Owasp,» 2007. [En línea]. Available: https://www.owasp.org/images/2/2b/Conferencia_OWASP.pdf. [Último acceso: 10 09 2014].

[17] N. Security, «Norton,» [En línea]. Available: http://es.norton.com/security_response/glossary/define.jsp?letter=s&word=smurf-dos-attack. [Último acceso: 12 09 2014].

[18] areshelpusersrhb, «areshelpusersrhb,» [En línea]. Available: http://areshelpusersrhb.blogspot.com/p/ataques-dos-syn-icmp-udp-flood.html.

[19] mcmaster, «wiki,» [En línea]. Available: http://wiki.cas.mcmaster.ca/index.php/IP_Spoofing.

[20] «Zona Viruz,» [En línea]. Available: http://www.zonavirus.com/articulos/que-es-el-spoofing.asp. [Último acceso: 20 09 2014].

[21] windowsecurity, «windowsecurity,» [En línea]. Available: http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part1.html.

[22] F. Catoira, «Welivesecurity,» 18 Junio 2012. [En línea]. Available: http://www.welivesecurity.com/la-es/2012/06/18/dns-spoofing/. [Último acceso: 30 Septiembre 2014].

[23] L. Hewlett-Packard Development Company, «Hewlett-Packard Development Company, L.P.,» [En línea]. Available: http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04123356. [Último acceso: 19 Junio 2015].

[24] Dlink, «Dlink,» [En línea]. Available: http://support.dlink.com/Emulators/di624_revC/h_wizard.html. [Último acceso: 19 Junio 2015].

[25] I. D-Link Systems, «D-Link Systems, Inc.,» [En línea]. Available: http://www.dlink.com/-/media/Consumer_Products/DI/DI_624M/Manual/DI624M_manual_en_us.pdf. [Último acceso: 19 Junio 2015].

[26] J. Reich, «frenetic,» frenetic, [En línea]. Available: https://bitbucket.org/reich/pyretic-vms/downloads/Pyretic-0.2.2-amd64.zip. [Último acceso: 19 Julio 2015].

[27] JetBrains, «Jetbrains,» [En línea]. Available: https://www.jetbrains.com/pycharm/. [Último acceso: 19 Junio 2015].

[28] S. Tatham, «chiark,» [En línea]. Available: http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe. [Último acceso: 19 Julio 2015].

[29] Xming, «straightrunning,» [En línea]. Available: http://sourceforge.net/projects/xming/files/latest/download. [Último acceso: 19 Julio 2015].

[30] ntsecurity, «ntsecurity,» [En línea]. Available: http://ntsecurity.nu/downloads/etherchange.exe. [Último acceso: 19 Julio 2015].

[31] A. Fulwood, 10 Diciembre 2013. [En línea]. Available: http://robospatula.blogspot.com/2013/12/man-in-the-middle-attack-arpspoof-sslstrip.html. [Último acceso: 14 Junio 2015].

[32] pentestmag, «pentestmag,» 14 Mayo 2014. [En línea]. Available: http://pentestmag.com/ettercap-tutorial-for-windows/. [Último acceso: 14 Junio 2015].

[33] S. Sanfilippo. [En línea]. Available: http://www.hping.org/manpage.html. [Último acceso: 14 Junio 2015].

[34] enlinux. [En línea]. Available: http://www.enlinux.org/uso-de-thc-ssl-dos-en-kali-linux/?print=pdf. [Último acceso: 14 Junio 2015].

[35] P. university. [En línea]. Available: https://www.cs.princeton.edu/~dpw/papers/frenetic-abstract-padl-2014.pdf. [Último acceso: 18 04 2015].

[36] Frenetic, «frenetic,» 10 2013. [En línea]. Available: http://frenetic-lang.org/publications/pyretic-login13.pdf. [Último acceso: 25 01 2015].

[37] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/tree/master/pyretic/core. [Último acceso: 18 04 2015|].

[38] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/blob/master/pyretic/core/classifier.py. [Último acceso: 18 04 2015].

[39] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/blob/master/pyretic/core/language.py. [Último acceso: 18 04 2015].

[40] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/blob/master/pyretic/core/network.py. [Último acceso: 18 04 2015].

[41] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/blob/master/pyretic/core/packet.py. [Último acceso: 18 04 2015].

[42] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/blob/master/pyretic/core/runtime.py. [Último acceso: 18 04 2015].

[43] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/wiki/Running-Pyretic. [Último acceso: 20 04 2015].

[44] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/wiki/Main-method. [Último acceso: 20 04 2015].

[45] J. Reich. [En línea]. Available: https://github.com/frenetic-lang/pyretic/wiki/Language-Basics. [Último acceso: 20 04 2015].

[46] 12 06 2012. [En línea]. Available: http://www.backtrack-linux.org/wiki/index.php/Main_Page. [Último acceso: 20 04 2015].

[47] [En línea]. Available: https://www.kali.org/. [Último acceso: 20 04 2015].

[48] J. Reich, «frenetic-lang,» 27 Febrero 2015. [En línea]. Available: https://github.com/frenetic-lang/pyretic/wiki/running-pyretic. [Último acceso: 19 Abril 2015].

ANEXOS

ANEXO A

El código de la aplicación se encuentra en el CD adjunto.