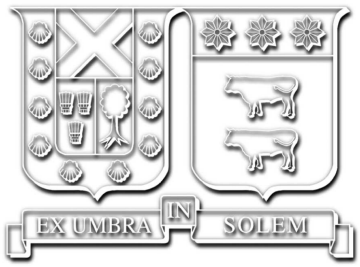
Universidad Técnica Federico Santa María
Departamento de Ingeniería Electrónica
Valparaíso, Chile



DISEÑO, OPTIMIZACIÓN E IMPLEMENTACIÓN DE UN SISTEMA DE DETECCIÓN DE INTRUSIONES HÍBRIDO

CÉSAR HERNÁN REYES PINO

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO MENCIÓN COMPUTADORES

Profesor Guía

Rudy Malonnek

Profesor Co-referente

Agustín González

Valparaíso, Julio 2016

RESUMEN

Los Sistemas Detectores de Intrusos deben formar parte de la seguridad de todo tipo de redes y sistemas informáticos. Actualmente, la mayoría de estos sistemas de detección se basan en firmas para localizar ataques ya conocidos. Este trabajo propone y desarrolla un prototipo de IDS híbrido que proporciona seguridad ante ataques conocidos y ante ataques nuevos. Para ello, usa un conocido IDS basado en firmas llamado Snort y propone un sistema de *grandes diferencias* para detectar anomalías. El sistema está diseñado para detectar trafico fuera de lo común usando información de la cabecera de los paquetes IP y por otro lado detectar trafico malicioso con firmas.

Palabras clave
Seguridad informática, Snort, firmas, anomalías, detección de intrusiones, código malicioso, sistema híbrido, patrones de comportamiento.

ABSTRACT
Intrusion Detection Systems must be included in security system of any networks or computer systems. Nowadays, the great majority of detection systems are based on signatures to find already known assaults. This work proposes and develops a prototype of hybrid IDS that provides safety with known and new assaults. To that effect, the system uses a known IDS based on signatures called Snort and proposes a system based on great difference to detect anomalies. This system is designed to detect outlier traffic using header information of IP packets and for other hand detect malicious traffic with signatures.

Keywords
computer security, Snort, signatures, anomalies, intrusions detection, malicious code, hybrid system, pattern of behaviour.

AGRADECIMIENTOS

A lo largo de todos estos años de Universidad vi, conocí y compartí con muchas personas maravillosas que me enseñaron cosas que no se aprenden en la sala de clase y que son primordiales en la formación de lo que soy ahora. Por eso le agradezco al Rocko, Chico, Mata, Seba, Maass, Cris y al Fito por todas esas noches de estudio y apoyo incondicional a lo largo de esta carrera. Al Marco, Christian, Chico, Aníbal y Paz por estar codo a codo trabajando por nuestros compañeros en el Centro de Alumnos, donde aprendimos sin duda a trabajar verdaderamente en equipo. A Jorge, Cristian, Rickemberg, Lesdy y Vanessa por encabezar junto el gran desafío de representar a los estudiantes en la FEUTFSM. A Felipe, Selma y Leonardo grandes compañeros de militancia que llevan por delante los mismo ideales transformadores que llevaran a una sociedad verdaderamente justa y democrática. Al Leo, Amaro, Duarte y Pipe por dedicarle su tiempo a trabajar por una Universidad diferente. A los desbozeros Guille, Javier, Christian, Paz, Nara, Thomas, Nico, Lucho, Seba que hicieron de una casa un hogar muy acogedor. A todos los funcionarios y profesores que ayudaron en mi formación personal y profesional.

A mis padres que siempre creyeron en mí y en mis capacidades, dejando crecer mis sueños. A mi hermana fuente de inspiración por el conocimiento y mi hermano que desde temprano se encantó por descubrir y conocer más allá de lo común.

Por último agradecer por todo el apoyo y aguante a Antonella para cerrar esta etapa de mi vida y seguir construyendo nuestra vida juntos.

CONTENIDO

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

La nueva tendencia respecto de la implementación de gran cantidad de dispositivos móviles y redes inalámbricas como parte de la nueva infraestructura tecnológica, ha requerido que los profesionales en seguridad, ajusten nuevamente sus procedimientos y desarrollen un conjunto de técnicas y controles capaces de velar por la seguridad de la información con ellos relacionada. En resumen, la implementación de nuevas tecnologías indefectiblemente trae aparejado, el advenimiento de nuevas oportunidades de negocio, así como también riesgos, amenazas y nuevos vectores de ataque, siendo requerido como parte de un proceso continuo, la revisión constante de los modelos de seguridad y la adecuación de controles, de modo tal de mantener su vigencia.

De esta manera la implementación de más niveles de seguridad toma importancia hoy en día, cuando el uso de las redes e Internet crece sin detenerse. Es en esa dirección en la que se enmarcar este trabajo de título que busca crear herramientas para minimizar los riesgos y exposición de la información del Departamento de Electrónica a través de un nuevo nivel de seguridad que es complementario a las herramientas que son usadas actualmente.

Los niveles de seguridad que se implementan actualmente en el Departamento de Electrónica son la primera barrera de protección, sin embargo hay servicios que deben estar accesibles y son punto vulnerables que frente a un ataque no pueden ser detectados en los términos actuales, es acá donde toman importancia los sistemas de detección de intrusiones los cuales son capaces de detectar intentos de vulneración a la seguridad. En particular este trabajo propone un Sistema de Detección de Intrusiones Híbrido, basado en detección por firmas a través del software libre Snort, y detección por comportamientos anómalos basado en un algoritmo llamado *Unsupervised Network Intrusion Detection System (UNIDS)* el cual es implementado como un preprocesador de Snort.

El orden de este trabajo parte con una investigación de la seguridad de la información, los tipos de ataques, las alternativas de solución a nivel de software y hardware, como

funciona y cuales son las partes fundamentales de Snort, la explicación del módulo nuevo para la detección anómala y los resultados obtenidos con el tráfico real de la red de electrónica.

Capítulo 2

SEGURIDAD INFORMÁTICA

Las instituciones, del tipo que sea, acumula información en diferentes dimensiones y con diferentes usos o fines. Gran parte de esta información es almacenada, tratada y puesta a disposición de sus usuarios en computadoras y transmitidas por la red con otros ordenadores. Parte de esta información puede ser confidencial o de acceso restringido, ya sea por motivos de soberanía nacional, información estratégica para una empresa o datos de acceso bancarios de los usuarios por ejemplo. Tomando así gran importancia el manejo y resguardo de esta información para que no caiga en manos equivocadas.

La Seguridad de la Información tiene como fin la protección del acceso, uso, divulgación, interrupción o destrucción no autorizada de la información y de los sistemas de la información.

Si bien los términos Seguridad de la información y seguridad informática se usan a menudo para referirse a lo mismo, no es lo mismo, el primero comprende un abanico más grande de posibilidades respecto a la segunda, donde esta última se centra más en los aspectos técnicos de los sistemas tecnológicos de la información. Sin embargo ambos buscan mantener la integridad, disponibilidad y la confiabilidad ya sea sea en papel, CD, correo electrónico, audio, etc.

La Seguridad de la información involucra la implementación de una estrategia que abarque distintos procesos y técnicas que permitan resguardar de forma primordial la información. Esta estrategia debe establecer políticas, controles de seguridad, tecnología y procedimientos para detectar amenazas que puedan explotar alguna vulnerabilidad que ponga en riesgo la información del sistema. Por lo tanto deben ser políticas que aseguren la protección de la información como de los sistemas que la almacenan.

Dado que la tecnología avanza constantemente, la seguridad debe estar en un proceso en continua mejora, por lo que las políticas y los controles establecidos deben revisarse y adecuarse a las nuevas necesidades a medida que sea necesario, con el fin de tomar acciones que permitan reducir o en el mejor de los casos eliminarlo todo intento de

vulneración de estas políticas de seguridad.

2.1. Pilares de la Seguridad de la Información

En el mundo de la Seguridad de la Información, por varios años viene declarando que la confidencialidad, integridad y disponibilidad [1] son los pilares fundamentales para mantener protegida la información. Cada uno de estos pilares se definen a continuación:

2.1.1. Confidencialidad

La confidencialidad se orienta a garantizar que la información del sistema solo sea accedida por usuarios autorizados. En casos donde esta es accedida por personal no autorizado puede generar grandes daños a las instituciones dado que la filtración de información confidencial puede dejar libre planes estratégicos de la misma o usar información privada de los usuarios con fines malévolos. Para evitar la violación de confidencialidad, se utilizan por ejemplo contraseñas y técnicas de encriptación.

2.1.2. Integridad

La Integridad de la Información es la característica que hace posible garantizar su exactitud y confiabilidad, velando porque su contenido permanezca inalterado a menos que sea modificado por personal autorizado, de modo autorizado y mediante procesos autorizados.
Una falla de integridad puede estar dada por anomalías en el hardware, software, procesos, virus informáticos y/o modificación por personas que se infiltran en el sistema, modificando o borrando los datos importantes que son parte del sistema de información sin tener autorización para ello.

El control de acceso, los sistemas de detección de intrusos, la aplicación de chequeo de integridad, los procedimientos de control de cambios, la separación de funciones y la implementación del principio de "Menor Privilegio", son solo algunos de los medios utilizados para prevenir problemas de integridad y en particular en el envío de mensajes se usan las firmas digitales para garantizar la integridad.

2.1.3. Disponibilidad

La disponibilidad, garantiza que la información del sistema se encuentre accesible en cualquier momento para algún usuario autorizado que desee acceder a ella. Esto requiere que la misma se mantenga correctamente almacenada, con el hardware y el software

[1] conocida cómo CIA por las siglas en inglés "Confidentiality, Integrity, Availability"

funcionando perfectamente y que se respeten los formatos para su recuperación en forma satisfactoria.

Entre las amenazas que afectan el principio de Disponibilidad, se encuentran las fallas relacionadas con el software y hardware, aspectos relacionados con el entorno (calor, frío, humedad, electricidad estática, etc.), desastres naturales, denegaciones de servicios (DoS, DDoS), etc.
A fin de prevenir inconvenientes que puedan afectar la Disponibilidad, deben ser implementados mecanismos de protección adecuados, con el fin de reforzar la estrategia de continuidad del servicio definida por la organización, previniendo de este modo amenazas internas o externas. En tal sentido deberían implementarse medidas de resguardo y recuperación, mecanismos redundantes, planes de contingencia, sistemas de prevención y/o detección de intrusos, procedimientos de hardening, etc. A su vez, puntos únicos de fallas deberían ser evitados.

También dentro de la literatura se suele usar un cuarto concepto dentro de los pilares de la seguridad, el Control o irrefutabilidad.

2.2. Tipos de ataques

La mayoría de los ataques informáticos solo corrompen la seguridad de los sistemas de una manera muy específica. Por ejemplo ciertos ataques pueden habilitar a un hacker leer un archivo específico, pero no permitir la modificación de cualquiera de los componentes del sistema. Otro ataque puede permitir a un hacker apagar ciertos componentes del sistema pero no tienen permiso para acceder a los archivos. A pesar de la variada capacidad de los ataques informáticos, estos usualmente resultan violar alguno de los pilares de la seguridad.

2.2.1. Tipos de ataques comúnmente detectado por los IDS

Si bien aún no entramos en la definición formal de un Sistema de Detección de Intrusiones (IDS), haremos algunas definiciones sobre los tipos de ataques más comunes en esta área lo que nos permitirá entender con mayor facilidad la lógica que busca estos sistemas.

Tres son los tipos de ataques más comunes reportados por los IDS:

- Scanning Attacks o Probing
- Denial of Dervice (DoS)
- System Penetration

Estos ataques pueden ser realizados localmente sobre la máquina atacada o remotamente utilizando una red para acceder a ella. Frente a ello es necesario revisar la diferencias entre estos tipos de ataques porque cada una requiere un conjunto diferente de respuestas.

2.2.2. Scanning Attacks/Probing

Un Scanning Attack se produce cuando un atacante explora una red o un sistema mediante el envío de diferentes tipos de paquetes. Usando la respuesta recibida desde el sistema objetivo, el atacante puede aprender mucho de las características y vulnerabilidades del sistema. Así un Scanning Atack actúa como una herramienta de identificación del sistema objetivos para un atacante, sin embargo este tipo de ataques no penetra o compromete la integridad del sistema.

Varios son los nombres de herramientas usadas para realizar estas actividades, alguna de ellas son: mapeo de la red, mapeo de puertos, exploración de red, exploración de puerto o exploración de vulnerabilidad.

Un Scanning Atack puede entregar:

- topología de la red objetivo.
- tipo de tráfico de la red permitido por el firewall.
- host activo en la red.
- sistema operativo que el host está corriendo.
- software del servidor que corre en el host.
- número de la versión todos los software detectados.

Las exploraciones de vulnerabilidad son un tipo especial de ataque que verifica vulnerabilidades específicas del host. Así, un hacker puede realizar una exploración de vulnerabilidad y obtener una lista de host que son vulnerables a un ataque específico. Con esta información un hacker puede identificar precisamente que sistemas son posibles víctimas en la red objetivo y que tipo de ataques puede realizar de manera efectiva para penetrar la seguridad del sistemas victima. De esta manera los hackers usan estos software antes del ataque real. Desafortunadamente para las víctimas, al igual que es legal para las personas entrar a un banco y analizar los sistemas de seguridad visibles, es legar para un hacker hacer un exploración a un host o una red. Porque desde esta perspectiva realizar una exploración es recorrer legalmente Internet en busqueda de recursos de acceso público.

Los motores de búsqueda de la web pueden explorar la Internet buscando nuevas páginas web. Un sujeto puede explorar en Internet en búsqueda de repositorios de música gratuita o para acceder a juegos multi-player públicos. Fundamentalmente, el mismo tipo de tecnología que permite descubrir los recursos disponibles para el público también permite analizar un sistema de fallas de seguridad.
Las mejores firmas de IDS para exploraciones maliciosas son usualmente capaces de discernir entre elementos legítimos o maliciosos. La exploración es probablemente el tipo de ataque más común, este es el precursor de cualquier intento de penetración serio. Si una red está conectada a Internet, lo más seguro es que fue explorada, si no diariamente, al menos un par de veces a la semana.

2.2.3. Denial of Service Attacks - DOS

Los ataques de denegación de servicio (Denial Of Service - DOS) intentan enlentecer o desactivar la red objetivo o el servicio ofrecido.
Son 2 los principales tipos de ataques DOS: explotación de las vulnerabilidades y defectos del sistema (Flaw exploitation) e Inundación (flooding). Es importante entender la diferencia entre ambas para saber cómo actuar frente a estas situaciones.

2.2.4. DOS Flaw exploitation

Los ataques Flaw exploitation tienen como objetivo hacer explotar alguna falla de los sistemas o software objetivo con el fin de provocar fallos en su procesamiento o hacer que se agoten los recursos del sistema. Un ejemplo de falla de procesamiento son los ataques 'ping of death'(detectado en 1997), estos implica enviar un paquete ping más grande de lo normal lo que generó fallos en los host ya que no podían procesarlos. En muchos casos un simple parche en el software puede evitar este tipo de ataques.

2.2.5. Ataques DOS Flooding

Los ataques de inundación son simplemente el enviar a un sistema o a un componente de este, más información del que este puede manejar. En los casos que el atacante no puede enviar suficientes paquetes para abrumar a su capacidad de un servidor y dejarlo inoperativo, el atacante puede sin embargo, ser capaz de monopolizar la conexión de red, negando así a cualquier otro usuario el acceso a ese servidor.
Este tipo de ataques no es tan fácil solucionar y si bien hay algunos mecanismo que ayudan a mitigar el problema, este sigue presente en la vida de los administradores de red. Así este tipo de ataques representa una gran fuente de frustración y preocupación de las organizaciones y si bien hay pocas soluciones para detener los ataques de inundación, hay varias modificaciones técnicas que se pueden hacer por un objetivo de mitigar

tal ataque.

El término de "distributed DOS" (DDOS) es un subconjunto de ataques DOS. En estos ataques el hacker usa múltiples computadoras para lanzar el ataque. Estos computadores atacantes son controlados centralizadamente por el hacker, logrando generar un ataque mucho más grande. Un hacker usualmente no puede derribar un gran sitio comercial de Internet por inundación con paquetes de la red de un solo host. Sin embargo si un hacker gana el control por ejemplo de 20.000 máquinas y logra hacer un ataque en una solo dirección al más rápido de los sistemas, la capacidad de realizar un ataque exitoso es muy probable.

2.2.6. Penetration Attacks

Los ataques de penetración adquisición y/o alteración intenten de manera desautorizada acceder a los privilegios, recursos o datos del sistema. Esta violación a la integridad y control del sistema contrasta con los ataques DOS que violan la disponibilidad de un recurso y los ataques de exploración que no hacen nada ilegal. Un ataque de penetración puede ganar el control de los sistemas por una explotación de varias fallas. Las fallas más comunes y las consecuencias de seguridad de cada una se enumeran a continuación. Si bien los ataques de penetración varían tremendamente en detalles e impacto, los tipos más comunes son:

- User to Root (U2R): El atacante tiene una cuenta en la máquina víctima e intenta ganar privilegios de super-usuario.
- Remote to Local (R2L): El atacante no tiene una cuenta en la máquina víctima e intenta acceder a ella.

2.3. Mecanismo de defensa

Para lograr proteger los sistemas informáticos de la inmensa cantidad de amenazas se han implementado diferentes métodos que permiten resguardar la información clave para las instituciones y usuarios.
El cifrado, la técnica más antigua de protección, la cual se basa en un conjunto de transformaciones de un espacio a otro donde cada cifrado usa llave particular.
El control de acceso, mecanismo usado por la mayoría de los sistemas es implementado como primera línea de defensa, limitando el acceso a un objeto en el sistema pero no restringe lo que el sujeto puede hacer con el objeto en caso de tener acceso a su manipulación.Si una contraseña es débil y se compromete, las medidas de control de acceso no pueden prevenir la pérdida o corrupción de la información a la que el usuario estaba autorizado a acceder.
Los mecanismos de identificación y autenticación de usuarios permiten la adecuada

identificación de los sujetos. La identificación busca conocer quien es el usuarios mientras que la autenticación, es la confirmación de su identidad permitiendo así mantener la integridad y la confidencialidad.
Por último existen mecanismos que velan por la disponibilidad de los sistemas. Así por ejemplo tenemos listas de acceso y firewall que filtra los datos dejando solo los datos autorizados.
Junto con todo estos mecanismo de defensa, las instituciones implementan una última línea de defensa con técnicas enfocadas a la detección de amenazas tales como los antivirus y los sistemas detección de intrusiones. La diferencia sustantiva entre estas técnicas de defensa en comparación a la anterior radican en que estas últimas permiten detectar intentos de romper la seguridad de los sistemas .

Capítulo 3

SISTEMA DE DETECCIÓN DE INTRUSIONES

Dado el incremento de las tecnologías de la de información y su nivel de complejidad también han aumentado las amenazas para estos sistemas. Así es como más organizaciones implementan infraestructura de seguridad adicional como son los IDS, siendo esta una herramienta en la línea de defensa que tiene como fin ayudar en descubrir, determinar e identificar uso desautorizado, duplicación, alteración y destrucción de información del sistema [2].

Los sistemas de detección de intrusiones (Intrusion detection systems - IDS) pueden ser sistemas de software o hardware que monitorean los eventos ocurridos en un sistema informático o red, analizando en búsqueda de actividad sospechosa. También podemos definir un IDS como detector de eventos que clasifica aquellos que son legítimos del sistema y por otro lado aquellos que son una intrusión al sistema, generando alarmas o notificaciones al administrador.

Una **intrusión** es una violación a las políticas de seguridad de un sistema informáticos o la materialización de una posible amenaza, donde se ven comprometida cualquiera de las pilares de los *seguridad informática* (confiabilidad, integridad o disponibilidad). Por lo general se habla de amenazas producidas externamente pero es sabido que muchas de estas vienen del mismo sistema informático.

3.1. Modelo de sistemas de detección de intrusiones

A pesar de que hay varias implementaciones de IDS, las más comunes comparten la estructura de módulo colector, análisis, almacenamiento y respuesta como se muestra en la figura (3.1) [3].

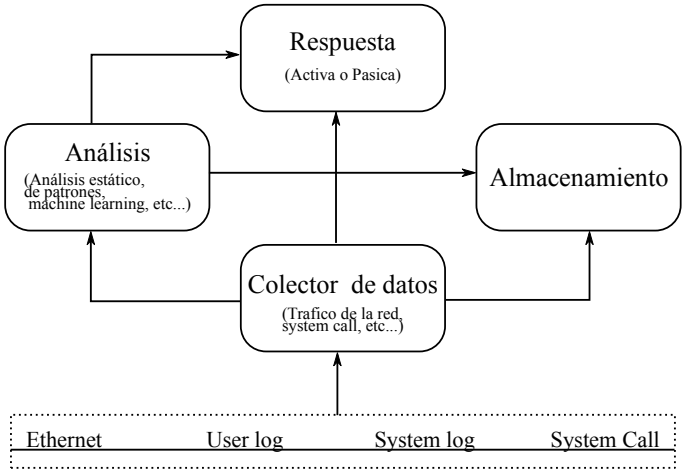


Figura 3.1: Estructura de funcionamiento de un IDS

Módulo Colector, tiene como objetivo revisar las fuentes o entradas que se van a analizar ya sea desde la red, de los archivos log o del sistema mismo. Lo importante de este módulo es que reduce la información a procesar.

Módulo Análisis, procesa los datos recolectados por el módulo colector. En muchas implementaciones este módulo se concentran en crear nuevas técnicas en términos del mejor rendimiento del IDS (clasificación rápida, reducir falsos positivo y de mayor ocurrencia), para ello se han desarrollado varias técnicas que van desde análisis estático, máquinas de patrones, machine learning, chequeo de integridad de archivos hasta sistemas inmunidad artificial. Lo importante de este módulo es ayudar a automatizar el análisis de datos para reducir la intervención humana y aumentar la velocidad de procesos de identificación de intrusiones en tiempo real.

Módulo de Almacenamiento, guarda información de los datos recolectados y del proceso de análisis para así poder crear nuevas política de seguridad con la información detectada, siendo ideal para hacer un análisis forense de los ataques posteriormente.

Módulo de Respuesta puede ser implementado como activo o proactivo, siendo estos últimos los más usados actualmente, estos generan una alarma cuando hay un intrusión para que el administrador tome acciones posteriormente. Por otro lado existen los Sistemas de Detección y Prevención de Intrusiones (IDPS) que no solo notifican si no que intervienen directamente en el ataque, bloqueando estos en tiempo real.

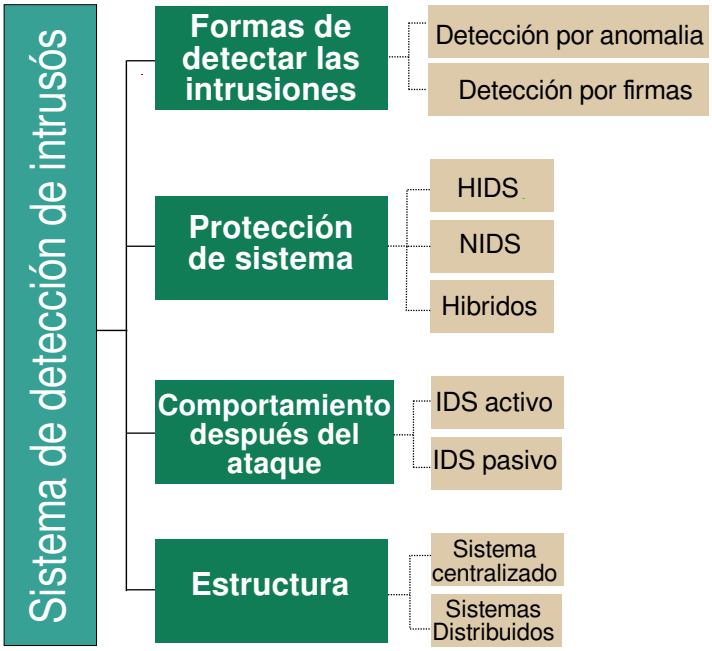


Figura 3.2: Tipos de clasificación en IDS

3.2. Tipos de detección de intrusión

Desde los inicios de los IDS hasta la actualidad se han desarrollado diferentes tipos de estrategias para diferentes enfoques en la detección de intrusiones, cada una de ella a evolucionado en el tiempo implementando diferentes algoritmos y formas de llevarlo a cabo, para ello existen una serie de clasificaciones que se agrupan en los siguientes ejes:

3.2.1. Clasificación por tipo de respuesta

Desde el punto de vista de las acciones que toman los IDS frente a los eventos de amenazas detectados podemos encontrar 2 grupos, activos o pasivos dependiendo de la injerencia en las acciones del sistema sobre la intrusión.

Respuesta pasiva

Las respuestas pasivas son la forma clásica de los IDS, en la cual se detecta un evento sobre un umbral de confianza y se registra la información y generando una alerta o notificación al administrador. Sin embargo no responde directamente a la amenaza, dejando así la responsabilidad de tomar acciones al administrador del sistema.
El enfoque de estos sistemas es la de creación de políticas después de ocurrida la amenaza de manera de evitar que vuelva a ocurrir en el futuro.

Respuesta activa

Las respuestas activas son propias de los IPS (Intrusion Prevention System) los cuales al detectar una amenaza actúan sobre la acción de manera inmediata deteniendo el ataque, evitando así una vulneración al sistema. Las acciones que toman estos sistemas dependiendo del enfoque del entorno analizado (red o host) pueden ser cortes de conexión o bloqueo de acciones dentro del SO.

3.2.2. Clasificación por tipo de entorno analizado

Dependiendo de donde provengan los datos a analizar, es donde se pondrán los sensores, dando así 2 tipos de clasificaciones ya sea por datos de la red Network IDS) o de datos propios de la máquina (Host IDS).

Host IDS - HIDS

Los HIDS analizan procesos del sistema, uso de CPU, acceso a ficheros, cuentas de usuario, registro de eventos, entre otros, generado por aplicaciones o por el mismo sistema operativo en búsqueda de posibles amenazas de usuarios o de procesos que quieran alterar alguno de los pilares de la seguridad de la información.

Network IDS - NIDS

Un NIDS analizan el tráfico de la red, esto significa que el objeto de análisis son los paquetes y tramas de los diversos protocolos de la misma. Estos eventos de red se dividen en dos partes: Header y Payload, en consecuencia se han desarrollado NIDS's que abordan el problema de la detección de intrusiones mediante el análisis de una de estas componentes dado que cada una de estas aproximaciones trae ventajas en la detección de distintos tipos de ataques.

Por un lado aquellas que se centran en los header de cualquiera de las capas de red, analizando por ejemplo los puertos origen destino o IP origen destino para generar un evento de análisis de amenaza, siendo muy rápido en la implementación, consumiendo muy pocos recursos. Por el otro lado aquellos que revisan el payload pueden hacer un análisis más profundo pero con un costo de recursos mayor.

3.2.3. Clasificación por tipo de detección

La clasificación por tipo de detección se basa en la cual es el criterio para decidir cuándo es o no una amenaza. Por un lado tenemos la detección basada en uso indebido o firmas, que usa un patrón conocido o firma como criterio y por otro lado están los sistemas

basados en anomalías los cuales usan el comportamiento normal del entorno monitorizado como patrón de detección. La mezcla de ambos enfoques da como resultado los sistemas híbridos de detección.

IDS basado en usos indebidos

Conocidos como *Signature Base System* (SBS) o *Misuse Base System* (MBS), los sistemas de detección por uso indebido se usan normalmente para detectar actividad que ocurren en el sistema comparando estas con una serie de firmas almacenadas previamente en una base de datos (DB) que se comparan con la actividad del entorno monitorizado, si hay coincidencia se notifica el evento con la información referente a la intrusión (funcionando similar a un antivirus). Esta forma es ampliamente usada en diferentes sistemas de de seguridad, existiendo un gran campo de desarrollo al respecto.

También existen técnicas donde se implementan maquinas de aprendizaje supervisado para generar modelos de los ataques para luego detectar los ataques.

Uno de los problemas de este enfoque radica en que el patrón o modelo debe ser conocido y estar en la DB, porque de lo contrario no se detectará, quedando totalmente vulnerable por el periodo de tiempo en el que se diagnostica un ataque y se crea una firma. Además pequeñas modificaciones en el ataque deja de ser detectado por el IDS siendo fácil de burlar. A pesar de esto es altamente preciso en la detección generando muy pocos falsos positivos.

IDS basado en detección de anomalías

Los sistemas basados en detección de anomalías o *Anomaly Base System* (ABS) se basan en crear perfiles de comportamiento normal del sistema. Luego de definir lo que es normal los ABS clasifican las actividades sospechosas que está fuera de los que se declaró normal, entonces genera una alerta.

Así en una primera fase los ABS usan métodos heurísticos y estadísticos (otras aproximaciones tratan de incorporar otras técnicas para realizar esta función) que describen el normal comportamiento del sistema (etapa de entrenamiento) para después comparar el tráfico normal y notificar cuando ocurran eventos que no están dentro de lo normal del sistema.

Su principal ventaja es poder notificar de intrusiones nuevas que no se tiene registro (zero-day detection) dado que no usan firmas. Sin embargo sus principales desventaja son la generación de altas tasas de falsos positivos, junto con el problema radicado en la fase de entrenamiento, donde una intrusión podría no ser detectada quedando como

un comportamiento normal imposible de detectar en el futuro [4] [5].

Según Axelsson los ABS se podrían clasificar en 2 grupos, aquellos que tiene un enfoque de auto-aprendizaje, que comienzan automáticamente a monitorear el sistema y en el camino van formando el comportamiento normal de la red, un aprendizaje en tiempo real y por otro lado un enfoque programado, donde debe aprender de forma manual lo que se considera un comportamiento normal por tener un usuario o alguna forma de función de "enseñanza" a través de entradas de información [6].

A pesar de tener usualmente un alto costo computacional, hoy recibe gran atención por la rápida detección de frente a los ataques nuevos.

Sistemas Híbridos

También se puede usar las ventajas de ambos enfoques y unirlos para crear un sistema mucho más completo. Las razones para esto son el alto índice de aciertos y bajo número de falsos positivos ante vulnerabilidades conocidas de los IDS basados en uso indebido y la protección frente a nuevos ataques de los IDS basados en la detección de anomalías.

3.2.4. Clasificación en función de su estructura

Esta clasificación se orienta a la composición en la red del IDS, pudiendo ser centralizada o distribuida.

Centralizadas

Se ejecuta en una sola máquina donde pasa todo el tráfico de la red controlando así toda la seguridad. Su ventaja está en la fácil implementación y configuración pero requiere de mayor capacidad de procesamiento de esta máquina a diferencia de los sistemas distribuidos.

Distribuidas

Propio de estructuras de red que tienen flujos de red muy grandes lo que no hace factible analizar todo el tráfico en un solo punto de la red sin degradar la calidad de los servicio, es por esto que se diseñó un sistema donde se disponen de diversos sensores en máquinas o puntos de red los cuales notifican a una unidad central la que se encarga de procesar y cruzar los eventos, logrando así tener un panorama amplio de lo que está ocurriendo en la totalidad de la red referente a posibles ataques.

El tener varios sensores distintos por toda la red permite ampliar la información de la

que se dispone para la detección de un incidente en el sistema. Esto permite producir además una única respuesta a intrusiones visibles desde varios puntos de la red. Este tipo de IDS monitoriza la actividad entre varias redes, teniendo una visión global de esta.

3.3. Criterios de Evaluación

En los IDS basados en uso indebido se analiza el tráfico de la red y se compara con unas firmas (rules). Si el tráfico coincide con la firma (p.e. dirección IP, puerto, datos del paquete, etc) entonces el paquete se considerará como ataque. Y en los IDS basados en anomalías se va analizando el tráfico de la red para ver si el comportamiento de los usuarios se clasifica como ataque.

En el momento que un IDS toma una decisión, éste puede tomarla bien o mal. Por lo tanto y tal como se muestra en la figura 3.3, existen cuatro posibles estados:

- Falso positivo (FP): También se conoce como falsa alarma y corresponde a tráfico inofensivo que se considera como ataque.
- Falso negativo (FN): Ataque que no detecta el IDS.
- Verdadero positivo (VP): Ataque detectado correctamente
- Verdadero negativo (VN): Evento inofensivo que se etiqueta como tráfico normal

Lógicamente, el objetivo del IDS es maximizar los aciertos (verdaderos negativos y verdaderos positivos) y minimizar el número de fallos del IDS (falsos positivos y falsos negativos).

Para evaluar de un modo formal la precisión de un IDS es necesario conocer la probabilidad de detectar un ataque y de emitir una falsa alarma. Conociendo ambos valores se puede obtener la curva ROC (Receiver Operating Characteristic).

Para obtener una curva ROC sólo son necesarias las razones de Verdaderos Positivos (VPR) y de falsos positivos (FPR). La VPR mide hasta qué punto un clasificador o prueba diagnóstica es capaz de detectar o clasificar los casos positivos correctamente, de entre todos los casos positivos disponibles durante la prueba. La FPR define cuántos resultados positivos son incorrectos de entre todos los casos negativos disponibles durante la prueba.

$$VPR = \frac{VP}{VP + FN})FRP = \frac{FP}{FP + VN} \tag{3.3.1}$$

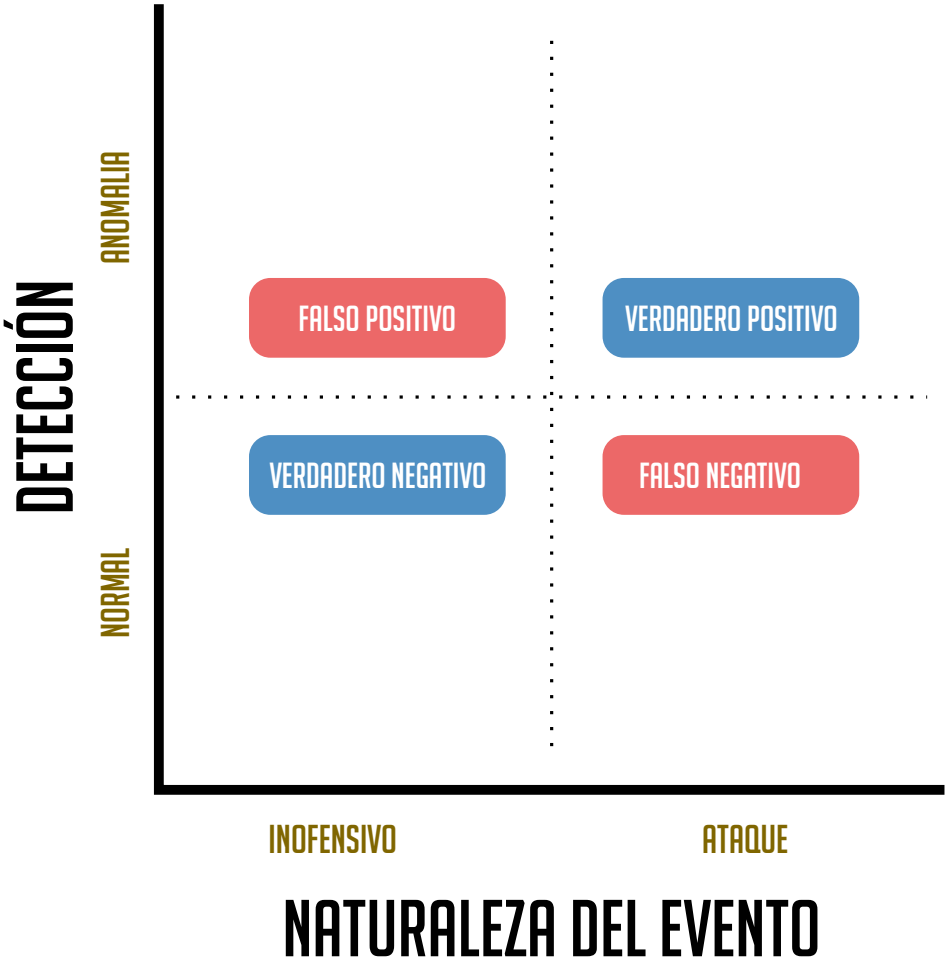


Figura 3.3: Criterios de Evaluación IDS

Si observa la curva ROC que se muestra en la figura 3.4, se puede ver que el IDS perfecto será aquel que detecte todo el tráfico de forma correcta y que no genere ninguna falsa alarma. Como este estado sería el ideal, el objetivo del proyecto es modificar un IDS para que tome decisiones maximizando el ratio de acierto y minimizando, en la manera de lo posible, las falsas alarmas que genera por ser un sistema híbrido.

3.4. Alternativas de solución

Teniendo alguna de las clasificaciones de los IDS y cruzando con los objetivos de este trabajo de título que busca implementar un nuevo sistema de seguridad para la Red de Computadores del Departamento de Electrónica que le brinde más herramientas al administrador sobre los que está pasando en la red es que se busca implementar un sistema híbrido de detección de intrusiones buscando aprovechar las ventajas de cada una

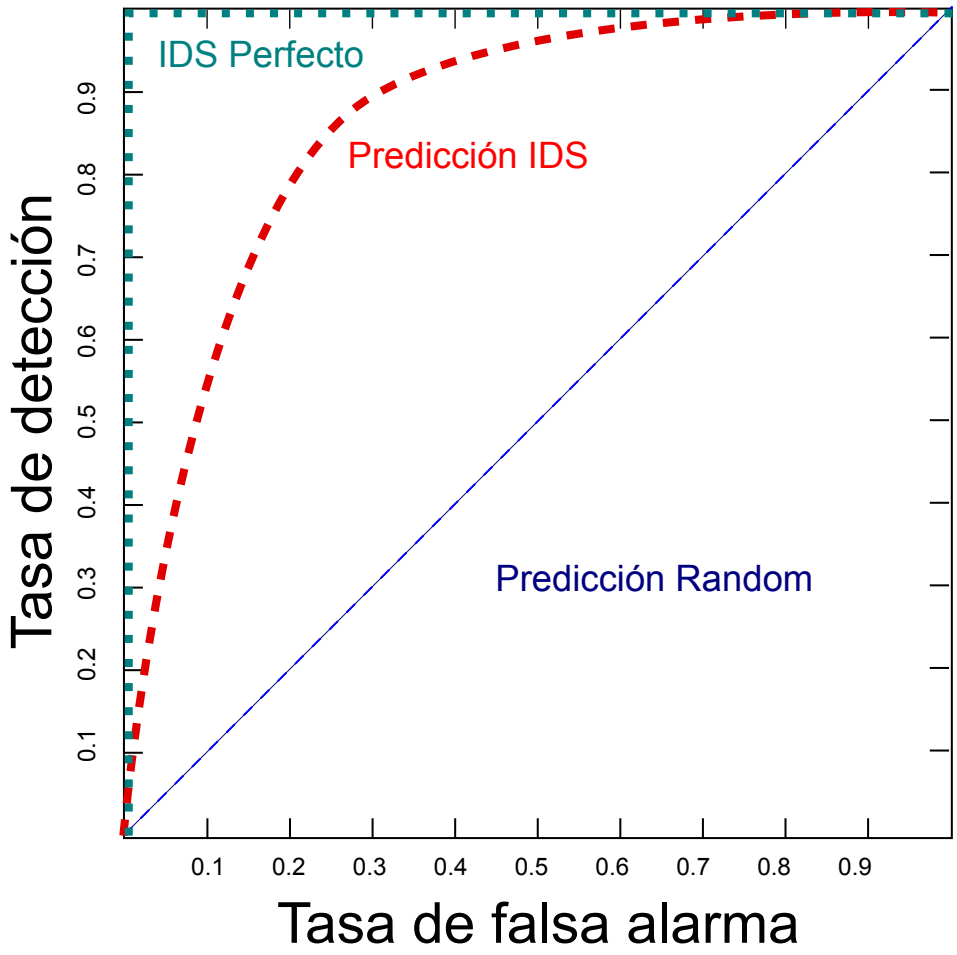


Figura 3.4: Curva ROC

de las tecnologías y tener un sistema capaz de reaccionar frente a ataques desconocidos junto con registrar todo tipo de actividad indebida con una base de datos de firmas.

Para la implementación de este sistema híbrido existen varias posibilidades, tanto por software como por hardware y para ello revisaremos alguna de ellas para tener un panorama general.

Para la solución por software se requiere unir los sistemas por separado dado que en la actualidad aún no hay software que esté desarrollado plenamente como híbrido, así es como usaremos alguno de los sistemas más conocidos en el área de la detección por uso indebido y se implementará en este un sistema basado en anomalías que en su mayoría solo están en investigación por lo que será un plugins de los primeros. Por otro lado tenemos las soluciones por hardware que son comerciales y tienen toda la estructura dedicada pero que no tiene posibilidades de desarrollo por lo que es difícil poder crear ideas nuevas en ellas.

3.4.1. Soluciones de software basadas en firmas

Tres son las principales alternativas de software libre multiplataforma que implementa sistemas de detección basadas en firmas, las que permiten perfectamente trabajar sobre ellas, a continuación un panorama de cada una:

Snort

Snort es un Sistema de Detección de Intrusiones de código abierto que es desarrollado por Sourcefire. Creado en 1998 por Martin Roesch y actualmente adquirido por Cisco desde el 2013, siendo esta la versión abierta de las solución comercial. Actualmente se encuentra disponible para diferentes SO (Linux, Mac OS, FreeBSD, UNIX y Windows) permitiendo una transversalidad en su uso y tanto la ingeniería de detección de Snort como la comunidad de reglas de Snort están bajo la licencia GNU GLP v.2 [7].

Snort está diseñando para analizar tráfico de la red en tiempo real o registros log si es que se quiera usar off-line.

Snort usa reglas producidas por SourceFire, la producidas por la misma comunidad (vulnerability research team - VRT) y también es posible que nosotros creemos nuestras propias reglas.

Suricata

Suricata es un sistema de detección y prevención de intrusiones de código abierto desarrollado por la Open Information Security Foundation (OISF) en el 2010, siendo muy nuevo en comparación a los otros disponibles. Según la OISF ellos no busca reemplazar o emular soluciones existentes sino al contrario buscan desarrollar una alternativa con ideas y tecnologías nuevas. El motor de Suricata y su biblioteca HTP están bajo la licencia GNU GPL v.2.
Actualmente Suricata está desarrollado para las plataformas Linux, FreeBSD, MacOS y Windows, dando transversalidad en el uso.

El modo de operación de Suricata es similar a la de Snort . Tiene básicamente la misma sintaxis de las reglas de Snort (casi 100 %) por lo que ambos sistemas pueden usar más o menos las mismas reglas.

Una de las ventajas significativas que tiene Suricata es su implementación del motor de de análisis multi-hebra lo que permite sacar el máximo provecho a las nuevas tec-

nologías de procesadores multinúcleo. Además de esto parte del equipo desarrolló una biblioteca HTP que es un analizador y normalizado del protocolo HTTP el que profundiza el análisis en este protocolo en búsqueda de nuevas amenazas.

Las reglas usadas por Suricata pueden ser las mismas que usar Snort (VRT) o Emerging Threats (ET), siendo estas últimas las más usadas por estar diseñadas especialmente para Suricata.

Bro

Bro es un analizador de tráfico de red pasivo de código abierto. Este es en primer lugar un monitor de seguridad que inspecciona todo el tráfico de la red en profundidad en búsqueda de signos de actividad sospechosa.

Desarrollado para sistemas basados en UNIX como Linux, FreeBSD y Mac OS bajo la licencia de BSD. Creado por Vern Paxson en 1998 en el Lawrence Berkeley National Laboratory (LBNL), actualmente cuenta con el soporte de la National Science Foundation (NSF) y avanzan en el desarrollo con International Computer Science Institute (ICSI).

El beneficio más inmediato es un extenso conjunto de archivos log que guardan la actividad de la red en un formato de alto nivel. Estos log incluyen no sólo un completo registro de cada conexión que se ven en el cable, sino también transcripciones de la capa de aplicaciones, tales como HTTP, MIME, DNS, SSL, SMTP y muchos más.

Se destaca en particular que Bro no es un sistema de detección de intrusiones basado en firmas clásica. A pesar de que es compatible con estas funcionalidades estándar, el lenguaje de scripting de Bro facilita un espectro mucho más amplio de enfoques en la búsqueda de actividad maliciosa, incluyendo la actividad la detección por firmas, detección por anomalías y análisis de comportamiento.

3.4.2. Solución de software basado en anomalías

Las soluciones basadas en anomalía tiene sustento desde hace varios años en investigaciones que se centran en el uso del data mining para la detección de anomalías. Dado que son varios los métodos, en esta sección se detalla de una manera ligera los principales métodos usados en la detección de intrusiones.

Una característica principal dentro del data mining es el paradigma de aprendizaje de los sistemas. En el **aprendizaje supervisado**, se realiza la estimación basada en un

conjunto de entrenamiento, del cual se conoce la clase a la que pertenece. Dicha estimación se extrae mediante la utilización de diversos algoritmos sobre un conjunto de registros de ataques previamente etiquetados. Se han utilizado una gran variedad de algoritmos, tanto por separado como en conjunto, para así poder aprovechar las capacidades de cada uno. Ejemplos de técnicas de este tipo son los árboles de decisión, reglas de asociación, etc.
En cambio, en el **aprendizaje no supervisado**, llamado también agrupación o clustering, el sistema de clasificación de patrones debe diseñarse partiendo de un conjunto de patrones de entrenamiento para los cuales no conocemos sus etiquetas de clase. Estas situaciones se presentan cuando no disponemos del conocimiento de un experto o bien cuando el etiquetado de cada muestra individual es impracticable. Dentro de este tipo de aprendizaje destacan algoritmos de redes neuronales, k-nearest neighbor, k-means, etc [8].

Support Vector Machines

Estas son versiones muy sofisticadas de los perceptrones. Donde un perceptrón permite muchos, en teoría infinitas, posibles separaciones de hiperplanos entre clases (clasificador), donde SVM produce un único hiperplano óptimo. Si un patrón no es separable linealmente en el espacio de características original, una SVM permite separar linealmente en un espacio de más dimensiones [9].

Probabilistic Model

Varios son los modelos usados en detección de intrusiones que permiten un buen resultado, como lo son las cadenas de Markov, modelos ocultos de Markov, también los modelos no markovianos como los clasificadores de Gauss y Naive Bayes y de igual forma los modelos estadísticos [9].
En particular las cadenas de Markov y los modelos ocultos de Markov (HMM ó Hidden Markov Models) son una secuencia de eventos, donde la probabilidad del resultado de un evento depende sólo del resultado del evento anterior. De esa misma manera, los HMM son una técnica probabilística para el estudio de series en el tiempo [8].

Decision Trees

Un árbol de decisión es un modelo de predicción utilizado en el ámbito de la inteligencia artificial. Dada una base de datos se construyen diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema. Muy usada en operaciones de búsqueda y clasificación.
Los árboles de decisión comienzan con una variable llamada nodo raíz el cual es dividido en 2 o más ramas, representaciones de clases separadas del nodo raíz (si es categórica) o de rangos específicos a lo largo de la escala del nodo (si es continua). Alguno de los

algoritmos que usan árboles de decisión para llevara cabo su labor son ID3, C4.5, C5.0, Random forest.

Clustering

Clustering es la tarea de asignar a un conjunto de objetos dentro de un grupo llamado **clusters** donde estos objetos tienen algo en común que los diferencia de otros grupos. Así es el caso de k-nearest neighbor, un algoritmo basado en aprendizaje de instancia (instance based learning), el cual toma los ejemplos de la etapa de entrenamiento y cuando llega una nueva instancia se buscan los objetos similares a él y los deja en la misma clasificación. La diferencia entre este tipo de métodos es la métrica usada para asignar la mayor proximidad a una clasificación, en este caso se pueden diferenciar medidas de distancias o densidades de los objetos.
Por otro lado están las esta K-means, una técnica de clustering basado en particiones donde se construyen k particiones de los datos donde cada partición representa un grupo o cluster. Estos métodos, crean una partición inicial e iteran hasta satisfacer el criterio de parada. Otros tipos de clustering son hierical clustering.

Bayesian Networks

Las redes Bayesianas son unas herramientas poderosas como modelos de decisión y razonamiento bajo incertidumbre [9]. Una versión simple de las Bayes Netwoks son las naive Bayes, las cuales son particularmente eficientes en tareas de inferencia. Estas se utiliza cuando queremos clasificar una instancia descrita por un conjunto de atributos en un conjunto finito de clases.

Softcomputing

Soft computing es un sistema adecuado para detección de anomalías cuando no se encuentran soluciones exactas. Generalmente se conocen una gama de métodos como Genetic Algorithms, Artificial Neural Networks, Fuzzy Sets, Ant Colony Algorithms y Artificial Immune Systems.

A continuación dejamos una pequeña descripción de algunos métodos:

Artificial Neural Networks Son modelos del comportamiento de las neuronas, las cuales interactúan entre ellas frente a diferentes estímulos y tienen una sola salida o resultado, así se componen de 3 partes, la capa de entrada, salida y la oculta que es donde se realizan las interacciones. Cada neurona tiene un peso el cual se ajusta en la fase de entrenamiento acorde al resultado esperado frente a una entrada conocida minimizando así el error.

Artificial Immune Systems Modela el comportamiento que tiene el sistema inmunológico del ser humano, donde este tiene como objetivo la defensa del organismo identificando lo que es propio y no propio (self/non-self) de él, de esta manera se busca garantizar los mecanismos que activen la respuesta frente a ataques o posibles ataques.

Genetic Algorithms son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación. Permitiendo de esta forma por ej. crear reglas básicas las cuales evolucionan y mutan acorde al ambiente donde se está desarrollando.

Combination learner methods

Estos métodos buscan combinar múltiples técnicas para obtener mejores resultados, así es como podemos encontrar técnicas que combinan una amplia variedad de clasificadores que den como resultado uno mejor que cada uno de ellos por separados, por otro lado encontramos técnicas que usan múltiples clasificadores que usan diversas fuentes con el objetivo de mejorar la precisión de la detección. Además hay enfoques que toman ideas de detección por firmas y las junta con ideas de detección de anomalías, la ideas es tomar las potencialidades de ambos enfoques.

De este último podemos encontrar a Zhang [10] que propone un marco sistemático donde se aplica algoritmos de Data Mining llamado Random Forests en conjunto con un sistema de firmas. Por otro lado Tong [11] propone un modelo de redes neuronales híbrido RBF/Elman que puede ser empleado para ambos sistemas de detección, este puede detectar eficientemente ataques puntuales o colaborativos porque tiene memoria de los eventos pasados.

Un inteligente sistema híbrido basado en redes neuronales es desarrollado por YU [12]. Este modelo es flexible, extensible a diferentes ambientes de red, mejora la precisión y el rendimiento de detección. Junto con este, Selim [13] propone un mejorar la tasa detección de ataques conocidos y desconocidos a través de múltiples niveles con redes neuronales y árboles de decisión.

Por último Casas [1], propone una mezcla de metodos de clasificación Sub-Space Clustering (SSC), Density-based Clustering y Evidence Accumulation Clustering (EAC) para lograr detectar ataques con gran precisión.

3.4.3. Soluciones basadas en Hardware

En esta sección se muestran algunas alternativas comerciales de hardware que implementen sistemas híbridos de detección de intrusiones para tener una visión que nos

permita comparar las soluciones existentes.

3.4.4. McAfee Network Security Platform

McAfee Network Security Platform es una solución de seguridad excepcionalmente inteligente que detecta y bloquea amenazas sofisticadas en la red. Gracias al empleo de técnicas de detección de amenazas avanzadas, va más allá de la comparación de patrones para ofrecer protección contra los ataques sigilosos con extrema precisión, mientras que su plataforma de hardware de nueva generación llega a alcanzar velocidades superiores a los 40 Gbits con un único dispositivo capaz de satisfacer las necesidades de las redes más exigentes [14].

Si bien no se detalla en profundidad las especificaciones, alguna de las características son:

- Detección de anomalías
- Compatibilidad con firmas de código abierto, definidas por McAfee y definidas por el usuario
- Detección heurística de bots
- Correlación de ataques

DoS Detección basada en umbrales y en análisis heurístico

DoS Detección basada en perfiles, autoaprendizaje

- Reputación de IP

3.4.5. Palo Alto Networks

Palo Alto Networks next-generation firewall, es una completa suite de seguridad la que cuenta con sistemas IPS híbridos que permiten detectar ataque conocidos como no conocidos.

Si bien no se detalla en profundidad las especificaciones, alguna de las características son [15]:

- Detección de anomalías basado en actividades fuera de lo establecido en los RFC de ciertos protocolos

- Detecta patrones de ataques a través de más de un paquete, teniendo en cuenta elementos como el orden de llegada y la secuencia.
- Detección de anomalías basada en estadísticas para prevenir ataques DoS
- Análisis heurísticos detecta paquetes de tráfico y patrones anómalos, tales como análisis de puertos.

3.4.6. Cisco

Next-Generation Intrusion Prevention System (NGIPS) es un conjunto de soluciones a un nuevo estándar para una protección avanzada ante amenaza, integrando conocimiento del contexto en tiempo real.

Si bien no se detalla en profundidad las especificaciones, alguna de las características son [16]:

- Creación de reglas personalizadas.
- Analizador de comportamiento de la red.
- Detección de tipo de archivo.

Capítulo 4

SNORT

Snort es un *Sistema de Detección y Prevención de Intrusiones (IPDS)* de código abierto basado en red. Capaz de analizar tráfico en tiempo real funcionando similar a un snniffer, monitoreando todo el tráfico de la red en búsqueda de posibles amenazas a los sistemas. Este utiliza reglas descriptivas para identificar qué tráfico debe ser analizado y un módulo de detección que permite registrar, alertar y responder a través de un módulo de salida, cualquier tipo de amenaza previamente definida en tiempo real. Snort realiza análisis de protocolo, contenido y también puede ser usado para detectar una variedad de ataques y exploraciones, como buffer overflows, escaneo de puertos, ataques CGI, SMB probes, DDOS, entre otras.

Snort es desarrollado por Sourcefire. Creado en 1998 por Martin Roesch y actualmente adquirido por Cisco desde el 2013, siendo esta la versión abierta de las solución comercial. Actualmente se encuentra disponible para diferentes SO (Linux, Mac OS, FreeBSD, UNIX y Windows) permitiendo una transversalidad en su uso, esto porque se definió usar un módulo colector externo (libcap) para obtener los paquetes de la red.

Tanto la ingeniería de detección de Snort como la comunidad de reglas de Snort están bajo la licencia GNU GLP v.2 [17].

Snort es una de las alternativas más usadas dentro de los IDS basado en firmas gracias a su gran base de datos de patrones de ataques conocidos y a la flexibilidad que presenta con sus preprocesadores para detectar anticipadamente intentos de ataque. Además las reglas o firmas están escritas en su lenguaje propio el cual es muy flexible, facilitando así la modificación de reglas existentes o la creación de reglas nuevas.

4.1. Arquitectura de Snort

A continuación se detalla la arquitectura y funcionamiento de Snort para la versión 2.9.7.3, la cual es utilizada en este trabajo de titulo. Si bien en el sitio web se encuentra la versión 3.0 que contempla bastantes mejoras aún está en su fase Alpha por lo que no es recomendable hacer uso de ella en sistemas reales de seguridad.

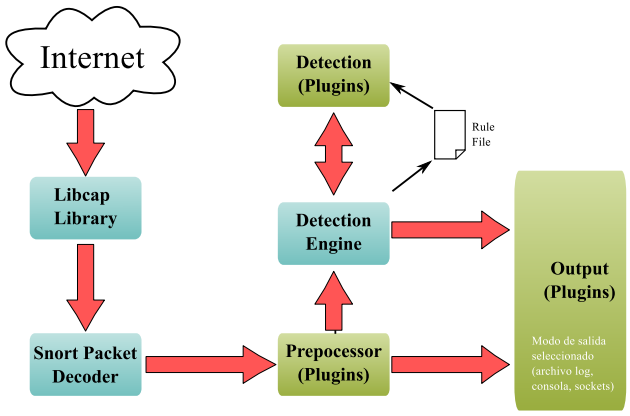


Figura 4.1: Estructura de Snort

La estructura de Snort se muestra en la fig (4.1), siendo los módulos Preprocessor, Detection y Output, plugins de Snort que pueden ser configurados individualmente dándole flexibilidad para la modificación y mejora de ellos.

En la figura (4.1) se muestra el flujo que desarrollan los datos de la red que Snort escuchar por la interfaz de red, donde los módulos cumplen funciones específicas los cuales se describen a continuación.

Packet capture library, en este módulo un software reúne los paquetes del adaptador de red usando *libpcap library* para capturar los datos de la red.

Packet decoder, toma los paquetes capturados y los decodifica, partiendo por la capa de enlace, luego la capa de red y luego la de transporte, de esta manera Snort tienen la información de cada uno de los protocolos para luego ser analizados.

Preprocessors, son plug-ins que operan en el decodificador de datos. El preprocessor fue introducido en la versión 1.5 y puede activar una alerta, clasificar o descartar un paquete antes de enviar esto al motor de detección que es más costoso en CPU. Por defecto Snort viene con una variedad de preprocesadores como Frag3, Sesion, Stream, sfPortscan, RPC Decode, Performance Monitor, HTTP Inspect, SMTP Preprocessor, POP Preprocessor, IMAP Preprocessor, FTP/Telnet Preprocessor, SSH, DNS, SSLTLS, ARP Spoof Preprocessor, DCE/RPC 2 Preprocessor, Sensitive Data Preprocessor, Normalizer, SIP Preprocessor, Reputation Preprocessor, GTP Decoder and Preprocessor, Modbus Preprocessor y DNP3 Preprocessor.

Detection engine, este es el módulo más importante de Snort, este opera en la capa de transporte y aplicación analizando el contenido de paquetes basado en la detección de reglas.

Detection Plugins, Partes del software que son compilados con Snort y se usan para modificar el motor de detección.

Output plugins, son una variedad de métodos de alerta y de log, activado por el preprocessor o una regla en el motor de detección. Snort registra en archivos de log en diferentes formatos como texto, binarios o syslog.

4.2. Packet capture library

Este módulo es el encargado de obtener todos el tráfico provenientes de la red en su forma completa, para entregárselo a los preprocesadores y luego el motor de detección.

Snort no usa un módulo propio para la captura, para ello usa por defecto una herramienta externa llamada *Libpcap* que le brinda los paquetes en formato RAW. Un paquete RAW tiene toda su información de la cabecera de protocolo de salida intacta e inalterada por el sistema operativo, a diferencia de las aplicaciones típicas que usan la información proporcionada por el SO, lo que permite tener toda la información que viaja por la red y así brindar un mayor campo de acción en la detección de ataques en contraposición a realizar la solicitud de los paquetes directamente al SO. Esto también ha permitido la portabilidad del sistema a otras plataformas dado que Libpcap esta en SO como MacOS, Linux, FreeBSD y Windows. No obstante desde la versión 2.9 se usa un módulo llamado DAQ (Data Acquisition library) para paquetes de entrada y salida que reemplaza los llamados directos a Libpcap con una capa abstracta que facilita las operaciones de varias interfaces hardware y software sin tener que modificar nada en Snort [17].

4.3. Packet decoder

El módulo de decodificación permite transformar el flujo de la red en estructuras de datos para los diferentes protocolos de la capa de enlace de datos, de red y transporte. De este modo Snort recibe diferentes paquetes de la red donde este módulo cuenta con diferentes decodificadores los que permiten identificar a qué clase de protocolo pertenece y así poder transformar esa información en un formato que permita a los siguientes módulos trabajar con la información en búsqueda de posibles amenazas o ataques. Un paquete recorre un flujo como el que se describe en la figura (4.2).

A través de los decodificadores, es que se debe alertar de diversos eventos como cabeceras truncadas u opciones de tamaños inusuales o no frecuentes en las opciones de TCP.

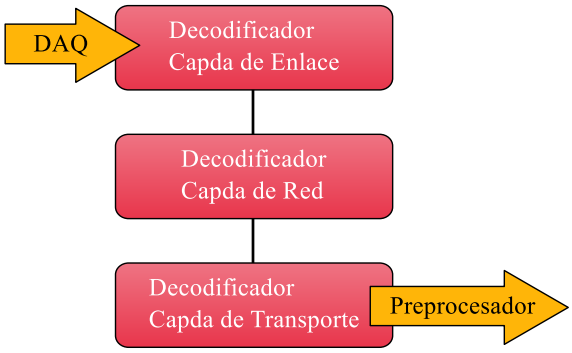


Figura 4.2: Flujo de datos decodificación Snort

4.4. Preprocessors

Son programas independiente de Snort, escritos en c, los cuales están justo después de la decodificación de los paquetes y justo antes del motor de detección. Estos programas cumplen funciones específicas que permiten el normalizado y la supervisión del tráfico de una manera más compleja que la que se realiza a través de las firmas. Esto ayuda a tomar la información de la red que viaja de una manera caótica y desordenada y darle forma coherente para luego interpretarla en los siguientes módulos, así por ej. toma todos los paquete que provienen de una misma ip y puerto y los ordena para luego analizar ya sea con un preprocesador o luego en el motor de detección con las firmas o reglas.

Estos pequeños módulos son compilados junto a Snort en forma de librerías. Una vez compilados la forma en la que Snort carga los módulos depende del orden en el que se encuentren en el archivo de configuración (**snort.conf**).

A continuación se dará un detalle de los preprocesadores con los que cuenta Snort.

4.4.1. Frag3

Este preprocesador es un módulo de Snort para la desfragmentación de los paquetes IP, este funciona bajo el concepto de IDS "target-based", esto significa analizar el tráfico de la red como lo haría el objetivo o el SO del host final. Cada SO implementa una forma diferente la desfragmentación, y se ha detectado que hay formas de evadir la detección de una amenaza a través de este método. Para ello se crea este módulo que permite calibrar la desfragmentación dependiendo del sistema que se esté defendiendo.

4.4.2. Session

Este preprocesador provee una API que permite la creación y la gestión de sesiones de control de bloques, capaz de seguir la pista de la sesión tanto para el protocolo TCP como para los protocolos UDP e ICMP, estos métodos son llamados a identificar las sesiones que podrían ser ignoradas (gran transferencia de datos, etc), y actualiza la información sobre la sesión (protocolo de aplicación, dirección, etc), las cuales pueden ser usadas después por reglas.

4.4.3. Stream

El preprocesador Stream es un módulo reensamblaje basado en objetivos para Snort. Este es capaz de rastrear sesiones tanto para TCP como para UDP e ICMP.
Stream como Frag3, introducen acciones basadas en objetivos para la manipulación de overlapping data y otras anomalías TCP.

Stream5 permite a otros protocolos normalizadores o preprocesadores configurar dinámicamente el reensamblaje de paquetes. Esta funcionalidad es utilizada por los protocolos de la capa de aplicación para identificar sesiones que podrían ser ignoradas y descargar información de identificación relativa a la sesión (protocolo de aplicación, dirección...) que puede ser usada posteriormente por las reglas del Detector Engine.

4.4.4. sfPortscan

El módulo sfPortscan, desarrollado por Sourcefire, es diseñado para descubrir la primera fase en un ataque de red: Scanning Attacks/Probing. En esta fase, un atacante determina qué tipos de protocolos de red o servicios soporta un host. Detecta escaneos a nivel de herramienta, como puede ser un escaneo de puertos realizado con Nmap. El preprocesador sfPortscan es capaz de detectar los siguientes ataques:

1. TCP/UDP/IP Portscan. El host atacante escanea múltiples puertos del host víctima.

2. TCP/UDP/IP Decoy Portscan. El atacante tiene un sistema que intenta camuflar su dirección IP con múltiples direcciones.

3. TCP/UDP/IP Distributed Portscan. Múltiples hosts atacantes escanean un único host víctima.

4. TCP/UDP/IP/ICMP Portswap. Un único host escanea un único puerto del host víctima. Esto suele ocurrir cuando un nuevo exploit sale a la luz y un atacante intenta aprovecharse de él.

SfPortscan también es capaz de detectar estos ataques cuando son filtrados (Filters Alerts). Esto ocurre cuando el host víctima no emite mensajes de error al host atacante (pero no porque no debería enviarlos, sino porque está configurado para no enviar nunca mensajes de error).

4.4.5. RPC decode

El preprocesador rpc decode normaliza múltiples registros RPC fragmentados en un único registro infragmentado. Si stream5 está habilitado, esto sólo tratará el tráfico del lado cliente. Por defecto, se ejecuta sobre los puertos 111 y 32771.

4.4.6. Performance Monitor

Este preprocesador mide el funcionamiento en tiempo real de Snort y teoriza su máximo rendimiento. Siempre que este preprocesador se activa, debería tener un modo de salida permitido, ya sea por consola, que imprime la estadística por pantalla o un archivo, donde el nombre se debe especificar, donde se almacenen las estadísticas.

4.4.7. HTTP Inspect

HTTP Inspect, es un decodificador genérico HTTP para usos de aplicaciones de usuario. Dado un buffer de datos, HTTP Inspect, decodifica el buffer, y normaliza los campos HTTP encontrados. HTTP Inspect trabaja tanto con respuestas del cliente como del servidor.

HTTP Inspect tiene una nutrida configuración para los usuario. Estos pueden configurar servidores HTTP individuales con una variedad de opciones, que debería permitir a los usuarios emular cualquier tipo de servidor web.
Dentro de HTTP Inspect hay 2 áreas de configuración: global y server.

4.4.8. SMTP Preprocessor

El preprocesador SMTP es un decodificador SMTP para aplicaciones de usuario. Considerando un buffer de datos, SMTP decodifica y encuentra comandos y respuestas SMTP.

4.4.9. POP Preprocessor

POP es un decodificador POP3 para el uso de aplicaciones de usuario. Considerando un buffer de datos, POP3 descodifica el buffer y encuentra órdenes POP3 y respuestas. También marcará comandos, datos de cabecera, datos de secciones y extraerá los

adjuntos de POP3, decodificando esto apropiadamente. Para que funciones POP Preprocessor, Stream Preprocessor debe estar activado.

Se debe asegurar que el puerto POP estén agregados a los puertos de stream5 para poder ser reensamblados.

4.4.10. IMAP Preprocessor

IMAP es un decodificador IMAP4 para el uso de aplicaciones de usuario. Considerando un buffer de datos, IMAP descodifica el buffer y encuentra órdenes IMAP4 y respuestas. También marcará comandos, datos de cabecera, datos de secciones y extraerá los adjuntos de IMAP4, decodificando esto apropiadamente.

Stream debe estar activado para que funciones IMAP. Se debe asegurar que el puerto IMAP estén agregados a los puertos de stream5 para poder ser reensamblados.

4.4.11. FTP/Telnet Preprocessor

FTP/Telnet es una mejora al decodificador Telnet y proporciona la capacidad de inspeccionar tanto FTP como Telnet. Este preprocesador decodifica el flujo, identificando comando y respuestas FTP, como secuencias de escape y campos normalizados Telnet. FTP/Telnet trabaja tanto con respuestas del cliente como del servidor.

FTP/Telnet tiene una nutrida configuración de usuario, similar a la de HTTP Inspect. Usuarios pueden configurar servidores y clientes FTP individuales con una variedad de opciones, que debería permitir a los usuarios emular cualquier tipo de servidor o cliente FTP. Dentro de FTP/Telnet hay 4 áreas de configuración: global, telnet, cliente FTP y servidor FTP.

4.4.12. SSH

El preprocesador SSH detecta los siguientes exploits: Challenge-Response Buffer Overflow, CRC 32, Secure CRT y el Protocol Mismatch.

Ambos ataques, Challenge-Response Overflow y CRC 32, ocurre después del intercambio de claves, y son por lo tanto encriptadas. Ambos ataques involucran el envío de un payload grande (20kb+) al servidor inmediatamente después del cambio de autenticación. Para detectar los ataques, este preprocesador cuenta el número de bytes transmitidos al servidor.

4.4.13. DNS

El preprocesador DNS decodifica respuestas DNS y puede detectar los siguientes ataques: DNS Client RData Overflow, Obsolete Record Types y Experimental Record Types.
DNS preprocesador mira la respuesta al tráfico DNS sobre UDP y TCP, esto requiere del preprocesador Stream para estar habilitar para la decodificación.

4.4.14. SSL/TLS

El tráfico encriptado debería ser ignorado por Snort por razones de rendimiento y para reducir falsos positivos. El SSL Dynamic Preprocessor (SSLPP) decodifica tráfico SSL y TLS y opcionalmente determina si Snort debería detenerse a inspeccionar dicho tráfico y cuando debería hacerlo.

Típicamente SSL es usado sobre el puerto 443 par HTTPS. Para habilitar SSLPP e inspeccionar el puerto 443 junto con activar la opción noinspect_encrypted, solo el handshake de cada conexión SSL es inspeccionada. Una vez que el tráfico es determinado como encriptado, no se inspeccionan los datos de la conexión.

Por defecto, SSLPP busca el handshake packet que sigue al tráfico encriptado que viaja en ambas direcciones. Si uno de los lados responde con una indicación de que algo ha fallado, entonces la sesión no es marcada como encriptada.

Para verificar que el tráfico encriptado enviado desde las dos partes es legítimo es necesario asegurarse de dos cosas: que el último handshake packet no fue manipulado para evadir a Snort y que el tráfico es legítimamente encriptado.

En la mayoría de los casos, especialmente cuando los paquetes pueden perderse la única respuesta que se observa del endpoint son los TCP ACK's.

4.4.15. ARP Spoof Preprocessor

El preprocesador ARP Spoof decodificá los paquetes ARP y detecta ataques ARP, solicitudes unicast ARP e inconsistencia en el mapeo Ethernet a IP.

Cuando los argumentos para ARP spoof no son especificados, el preprocesador inspecciona la dirección Ethernet y la dirección en el paquete ARP. Cuando ocurre una inconsistencia se genera la alerta.

4.4.16. DCE/RPC 2 Preprocessor

El principal propósito de este preprocesador es realizar la desegmentación SMB y desfragmentación de DCE/RPC para evitar evasión de reglas usando estas técnicas. La desegmentación SMB es realizada por las siguientes comandos que pueden ser usados para llevar peticiones y respuestas DCE/RPC: Write, Write Block Raw, Write and Close, Write AndX, Transaction, Transaction Secondary, Read, Read Block Raw y Read AndX.
Los siguientes protocolos de transporte están soportados por DCE/RPC: SMB, TCP, UDP y RPC sobre HTTP v.1 proxy y server.

4.4.17. Sensitive Data Preprocessor

El preprocesador de datos sensible, es un módulo de Snort que mejora la detección y filtrado de Información Personal Identificable (Personally Identifiable Information - PII). Esta información incluye número de tarjetas de crédito, número de seguro social (US) y direcciones email. Este es preprocesador requiere del preprocesador Stream para poder trabajar.

4.4.18. Normalizer

Cuando Snort opera en modo inline, este es útil para normalizar paquetes que ayudan a minimizar las posibilidades de evasión.

4.4.19. SIP Preprocessor

Session Initiation Protocol (SIP) es un protocolo de control (señalización) para crear, modificar y terminar sesiones con uno o más participantes. Estas sesiones incluyen llamadas telefónicas por internet, distribución y conferencias multimedia. El preprocesador SIP proporcionan maneras de abordar vulnerabilidades y explosiones comunes relacionados con SIP encontrados en los años anteriores. Esto también permite detectar nuevos ataques fácilmente. Además este requiere de los preprocesadores Stream y Frag3 para su funcionamiento.

4.4.20. Reputation Preprocessor

El preprocesador Reputation proporciona capacidades básicas de blacklist/whitelist IP, para bloquear/soltar/pasar tráfico de una lista de direcciones IP. En el pasado Snort implementó reglas para el bloqueo de IP basado en Reputation. Este procesador agrega una mejora que hace más fácil la gestión de la IP reputation. Este preprocesador corre antes que otros preprocesadores.

4.4.21. AppId Preprocessor

Con el incremento de la complejidad de las redes y el creciente tráfico de la red, los administradores de red requieren aplicaciones conscientes de la gestión de redes. Un administrador podría permitir solo aplicaciones que son relevantes para la institución, bajo ancho de banda y/o ocuparse de ciertas materias.

AppId agrega una visión en la capa de aplicaciones la gestión de la red. Con esto se agregan las siguientes características:

1. Network control: El preprocesador proporciona control de aplicaciones de punto único simplificado, haciendo un conjunto de identificadores de aplicación (App id) disponibles para escribir reglas para Snort.

2. Network usage awareness: las estadísticas de salida del preprocesador muestran el ancho de banda usado por la red por cada aplicación vista en la red. Los administradores pueden monitorear el ancho de banda usado y quizá decidir bloquear aquellas aplicaciones que son hacen uso excesivo.

3. Custom applications: el preprocesador permite a los administradores crear sus propios detectores de aplicaciones para detectar nuevas aplicaciones. Los detectores son escritos en Lua y se comunica con Snort usando una API definida en C-Lua.

4.4.22. GTP Decoder and Preprocessor

GTP (GPRS Tunneling Protocol) es usado en núcleo de las redes de comunicación para establecer un canal entre GSN (GPRS Serving Node). Este preprocesador provee formas para abordar intentos de intrusiones a través de redes GTP. Esto hace más fácil la detección de nuevos ataques.

4.4.23. Modbus Preprocessor

El preprocesador Modbus es un módulo de Snort que decodifica el protocolos Modbus, este protocolo es usado en la industria para controlar principalmente PLCs. Este preprocesador provee opciones de reglas para el acceso a determinados campos del protocolo. Esto permite al usuario escribir reglas para paquetes Modbus sin decodificar el protocolo con una serie de “content” y “byte_test”. Si en la red no se cuenta con este protocolo es recomendable desactivar este módulo.

4.4.24. DNP3 Preprocessor

El preprocesar DNP3 es un módulo Snort que decodifica el protocolo DNP3 (Distributed Network Protocol, en su versión 3), este es un protocolo industrial para comunicaciones entre equipos inteligentes (IED) y estaciones controladores, componentes de sistemas SCADA. Esto permite al usuario escribir reglas para paquetes DNP3 sin decodificar el protocolo con una serie de "content" y "byte_test".

4.5. Detection engine

El núcleo del funcionamiento de Snort es su motor de detección el cual tiene el objetivo de detectar, a través de diferentes reglas preestablecidas cualquier intento de intrusión o amenaza al sistema que provenga de un paquete de la red.

Las reglas son leídas en estructuras de datos que permiten comparar los paquetes provenientes de la red con estas. Si la comparación es coincidente entonces el sistema realiza una una acción (registrar o generar una alarma), de lo contrario el motor descarta el paquete.
El motor de detección tiene además plugins que permiten modificar las funcionalidades del motor de detección.

El motor de detección puede aplicar las reglas en diferentes partes de un paquete priorizando acorde a la complejidad de estas.

- **Reglas de cabecera IP**. Puede aplicar las reglas a las cabeceras IP del paquete.
- **Reglas de cabecera de la capa de Transporte**. Incluye las cabeceras TCP, UDP e ICMP.
- **Reglas de cabecera del nivel de la capa de Aplicación**. Incluye cabeceras DNS, FTP, SNMP y SMPT.
- **Reglas de contenido o payload**. Esto significa que se puede crear una regla que el motor de detección use para encontrar una cadena que esté presente dentro del paquete.

Pueden ocurrir múltiples detecciones pero la que se reporta es la de mayor prioridad.

Es importante tener en cuenta al momento de implementar Snort, que este módulo es el punto crítico y que podría eventualmente (en situaciones de alto tráfico) verse afectado los tiempos de respuesta o simplemente descartando paquetes lo que deteriora la detección en tiempo real y la calidad de la detección.

4.6. Output plugins

El módulo o plugins de salida es el que permite guardar o registrar las alertas generadas por el motor de detección o por los preprocesadores y que pueden tener diferentes estructuras dependiendo de las necesidades requeridas, controlando de esta forma la estructura y el tipo de información que se registra por el sistema.

Snort maneja varios formatos de salida como archivos log, mensajes a Syslog, Database, XML, modificación de configuración de routers y firewalls, además de Unified2 formato binario que sirve para que otro programa lo lea. El directorio que por defecto contiene los ficheros es *varlogsnort* donde todos los plugins activados dejan los registros. A continuación se muestra en detalle todos los plugins que maneja actualmente Snort.

- **Syslog**. Envía las alarmas al syslog muy fácilmente (basta con poner en la opción *-s* en línea de comando).
- **Alert_Fast**. El modo Alerta Rápida imprimirá en formato de 1 línea a un archivo determinado. Entrega información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen y destino.
- **Alert_Full**. El modo de Alerta Completa nos devolverá información completa incluyendo las cabeceras de los paquetes. Entregando información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, IP y puerto de origen/-destino e información completa de las cabeceras de los paquetes registrados. Este formato puede crear ficheros por IP del generador de la alarma. Además hay que considerar que este formato esta contemplado para redes de bajo tráfico.
- **Alert_unixsock**. Manda las alertas a través de un socket, para que las escuche otra aplicación/proceso en tiempo real.
- **Log_tcpdump**. Este módulo asocia paquetes a un archivo con formato tcpdump con el objetivo se usar otras aplicaciones para hacer post-procesamiento.
- **CSV**. El plugin de salida CSV permite escribir datos de alerta en un formato fácilmente importable a una base de datos.
- **Unified2**. Puede trabajar en uno de tres modos, packet logging, alert logging o true unified logging. Packet logging incluye una captura del paquete completo, alert logging solo registra la información del evento y para registrar ambas cosas en un solo archivo se especifica simplemente con unified2.
- **Log Null**. A veces es útil ser capaz de crear las reglas que provocarán alertas sobre ciertos tipos de tráfico, pero no causarán entradas en los archivos de log.

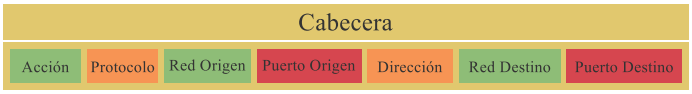


Figura 4.3: Estructura cabecera en reglas Snort

- **Log limit**. Esta sección se refiere a los log producidos por alert_fast, alert_full, alert_csv, log_tcpdump y tambien unified2 los cuales pueden tener límites de tamaño. Estos límites son descritos en sus respectivas secciones y cuando el límite es alcanzado, el actual log se cierra y se crea uno nuevo con una marca de tiempo.

Los archivos de registro pueden ser analizados de manera más rápida y resumida a través de Dashboard como ACID, SnortSnarf, Barnyard, SGUIL, Snorby o Kibana los cuales son administradores de las alarmas permitiendo la visualización y clasificación en tiempo real de lo que está detectando Snort.

4.7. Reglas/firmas

La reglas o firmas son patrones que se buscan dentro de paquetes que fluyen por la red en búsqueda de amenazas para el sistema. Estas reglas fueron diseñadas con un lenguaje ligero y muy potente además de ser libres, lo que permite adecuarlas a las necesidades particulares de cada sistema junto con crear reglas propias.

Snort usa reglas producidas por SourceFire, la producidas por la misma comunidad (vulnerability research team - VRT) y también es posible que nosotros creemos nuestras propias reglas.

4.7.1. Estructura de las reglas

Las reglas de Snort se dividen en 2 secciones lógicas, la **cabecera** y las **opciones**. En la cabecera se establecen las acciones que la regla va a ejecutar al momento de la detección, los criterios necesarios para que la regla identifique los paquetes de datos que se ajustan a ella (ver figura (4.3)) y la segunda sección, las opciones contiene mensajes de alerta e información de cada parte de los paquetes inspecciones y determinar si la acción de la regla se toma o no.

Cabecera de una regla

La estructura general de la cabecera de la regla es la que se puede observar en la tabla (4.1), a continuación se explica cada uno de los campos.

Estructura de la Cabecera de una regla Snort						
Acción	Protocolo	Red Origen	Puerto Origen	Dirección	Red Destino	Puerto Destino
alert	tcp	$EXTERNAL_NET	any		$HOME_NET	53

Tabla 4.1: Formato cabecera de regla Snort

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> \
              [192.168.1.0/24,10.1.1.0/24] 111 \
               (content:"|00 01 86 a5|"; msg:"external mountd access";)
```

Figura 4.4: Ejemplo de regla con lista de direcciones IP

- **Protocolo**. Permite establecer el protocolo de comunicaciones que se va a utilizar. Los posibles valores son: TCP, UDP, IP e ICMP.
- **Red de origen y red de destino**. Permite establecer el origen y el destino de la comunicación a través de una dirección IP y un bloque CIDR. También puede ingresar una lista de direcciones IP (figura (4.4)) y al usar '*!*' para permitir todas las IP menos una o un grupo de específicas de ellas. (figura (4.5)).
- **Puerto de origen y destino**. Permite establecer los puertos origen y destino de la comunicación. Indica el número de puerto o el rango de puertos aplicado a la dirección de red que le precede. Puede ser rango de puertos específico como por ej. 1:1024, :1024 (todos los menores), 1024: (todos los mayores) y además se puede usar el '!' para excluir un puerto de un rango o usar '*any*' para permitir todos.
- **Dirección**. Permite establecer el sentido de la comunicación. Existen 2 posibilidades, '-¿', lo que está al lado derecho del operador es el IP-puerto origen y a la izquierda IP-puerto destino, además existe la posibilidad de bidireccional '¡¿'.
- **Acción**. Permite indicar la acción que se debe realizar sobre dicho paquete. Los posibles valores son:
  - **alert**: Genera una alerta usando el método de alerta seleccionado y posteriormente registra el paquete.
  - **log**: Comprueba el paquete.
  - **pass**: Ignora el paquete.

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \
               (content:"|00 01 86 a5|"; msg:"external mountd access";)
```

Figura 4.5: Ejemplo de regla con negación de dirección IP

- **activate**: Alerta y luego activa otra regla dinámica.
- **dynamic**: Permanece ocioso hasta que se active una regla (basada en activate), entonces actúa como un inspector de reglas.
- **drop**: hace que iptables rechace y registre el paquete
- **reject**: iptables rechaza, registra y envía un TCP reset o un mensaje ICMP port unreachable, dependiendo del protocolo
- **sdrop**: iptables rechaza el paquete pero sin registrarlo
- Acciones definidas por el usuario
- Activate/Dynamic

Opciones de una regla

Las opciones, son los criterios que se establecen de manera adicional para encontrar los paquetes que puedan causar una amenaza acorde a lo que este escrito en la regla. Estas opciones se escriben dentro de un paréntesis y cada una de estas opciones van separadas por un ';', además cada una de estas opciones tiene 2 partes: keyword y argumento separadas por ':'.

Las 4 categorías principales de las opciones de reglas son:

1. **General**. Proporciona la información sobre la regla pero no tenga alguno afecta durante la detección. Alguna de ellas son msg, reference, sid, rev, classtype, priority, etc.
2. **Payload**. Busca patrones dentro de la carga útil del paquete (payload) y pueden estar interrelacionadas. Alguna de ellas son content, protected_content, hash, length, nocase, rawbytes, depth, offset, distance, within, etc.
3. **Non-Payload**. Busca patrones dentro de los demás campos del paquete, que no sean carga útil (por ejemplo, la cabecera). Alguno de ellos son fragoffset, ttl, tos, id, ipopts, fragbits, dsize, flags, flow, flowbits, seq, ack, etc.
4. **Post-Detection**. Permite activar reglas específicas que ocurren después de que se ejecute una regla. Alguno de ellos son logto, session, resp, react, tag, activates, etc.

A continuación se detalla alguno de los campos usado en las opciones de las reglas de Snort, para más detalle revisar en el manual de Snort [17].

- **msg**. Informa al motor de alerta el mensaje que debe imprimir cuando una regla se activa. Ante caracteres especiales como ':' y ';' se debe anteponer un \.

```
alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)
alert tcp any any -> any 80 (content:!"GET";)
```

Figura 4.6: Ejemplo de regla con la opción "content:"

- **referente**. Define un enlace a sistemas de identificación de ataques externos, como bugtraq, con id 788, proporcionando información sobre cómo se origina el ataque.
- **classtype**. Esta opción es usada para categorizar las reglas de detección de ataques en un gran conjunto de tipos de ataques. Esto permite una mejor organización de los eventos que Snort genera. Lo diferentes tipos que Snort maneja por defectos están contenidos en el archivo *classification.config* y tienen una sintaxis del tipo

  `<nombre_clase>, <descripción_clase >, <prioridad_por_defecto >`

  donde la prioridad es un valor entero, normalmente 1 para prioridad alta, 2 para media y 3 para baja.
- **sid**. Es usada únicamente identifica una regla Snort, permitiendo a los plugins de salida identificar las reglas más fácilmente.
- **content**. Es una de las características más importante de Snort, permitiendo a los usuarios configurar reglas para buscar de forma específica contenido en el payload y gatillar una alerta. Este puede contener una mezcla de texto y datos binarios (usualmente escrito en bytecode) como se aprecia en la figura (4.6).

Configuración

Las reglas de Snort se escriben en una sola línea o en varias líneas con un \al final de cada línea, las reglas están agrupadas en diferentes ficheros con la extensión **.rules* dependiendo del tipo de regla al cual pertenece. Todas las reglas por defecto se almacenan en el directorio /etc/snort/rules/.
El archivo de configuración *snort.conf* permite activar o desactivar grupos completos de reglas si es que el administrador lo requiere o si se desea desactivar una en particular hay que recurrir al archivo que contiene dicha regla.

4.8. Modos de Ejecución

Snort cuenta con 4 modos de ejecución distintos:

- Sniffer Mode.

- Packet Logger Mode.
- Network Intrusion Detection System (NIDS) Mode.
- Inline Mode.

A continuación detallamos los modos de uso de cada uno de ellos.

4.8.1. Sniffer Mode

Podemos usar Snort como un sniffer del trafico que pasa por la red. Con este modo no se procesa nada, solo recoge lo que pasa por la red y lo muestra directamente por la consola.
Algunas formas en la que podemos usar este modo se describen a continuación.

```
snort -v
```

El modo muestra por consola las cabeceras de los paquetes TCP/IP.

```
snort -vd
```

Este modo muestra por consola las cabeceras de los paquetes TCP/IP, UDP, ICMP.

4.8.2. Packet Logger Mode

El modo Packet Logger es similar al modo Sniffer con la diferencia de que guarda los paquetes capturados en la red en el disco a diferencia del otro que solo los muestra por la consola de al red.

```
snort -dev -l ./log
```

Este modo registra los paquetes en un archivo dentro de la carpeta especifica, la cual debe existir de lo contrario arroja un error.
Además hay otras opciones que permiten filtrar los paquetes los cuales se detallan ahora.

Opción	definición
-h	a continuación va una dirección IP especifica o una subred a capturar.
-b	registra los datos en un archivo binario, el cual puede ser usado por ser leído por otras aplicaciones como por ej. TCPdump. Para esta opción no es necesario poner -h ni -dev porque guardara todo.
-i	elige la interfaz a capturar
-r	a continuación el archivo que contienen datos registrados anteriormente (ej. packet.log) para poder ver lo por pantalla.

Tabla 4.2: Opciones filtro para Packet Logger Mode

4.8.3. Network Intrusion Detection System (NIDS) Mode

Es es el modo en el cual Snort funciona, como un IDS, captura los paquetes de la red, los analiza y compara con reglas, generando alertas sobre eventos sospechosos que están registrados en reglas predefinidas. La forma clásica para correr Snort es con la opción *-c* que permite especificar donde esté el archivo de configuración y la opción *-D* que deja corriendo Snort en modo servicio.

```
snort -c PATH/etc/snort/snort.conf -D
```

Junto con esta forma básica de ejecutar Snort en modo NIDS podemos sumar las opciones de Packet Logger Mode y Sniffer Mode. En el archivo *snort.conf* están todos los parámetros de configuración por defecto, si de manera particular se quiere modificar y no usar las opciones en la línea de comando cuando se corre Snort, solo hay que sustituir los valores por defecto. Un ejemplo de esto lo vemos en este ejemplo:

```
snort -dev -l ./log -h 192.168.4.0/24 -c ../etc/snort.conf -D
```

Además podemos cambiar la forma en la que se registran las alertas, esto se hace con la opción *-A* el cual permite cambiar entre

- Syslog. -A console
- Alert_Fast. -A fast
- Alert_Full. -A full
- Alert_unixsock. -A unsock
- Log_tcpdump.

- CSV.
- Unified2.
- Log Null. -A none
- Log limit.

4.8.4. Inline Mode

El modo In-Line es un plugins que tiene Snort para funcionar como un Sistema de Prevención de Intrusiones (IPS) que va más allá de solo notificar o generar alarmas, este modo puede bloquear o cerrar conexiones, funciona de manera activa frente a intentos de vulneración del sistema.
En este mode, Snort Obtiene los paquetes de los IPtables del sistema y usa las reglas de del IDS para determinar los paquetes que pueden entrar y salir del sistema.
Para el uso de esta opción es necesario compilar Snort con la opción de IPS

```
./configure --enable-inline
make
make install
```

Ahora hay que configurar el firewall para compatibilizarlo con este modo de Snort

```
iptables -A OUTPUT -p tcp --dport 80 -j QUEUE
```

Por último para ejecutar este modo podemos correrlo de la siguiente forma

```
snort_inline -QDc ../etc/drop.conf -l /var/log/snort
```

Las acciones que se tome en el modo Inline depende de las acciones que estén escritas en las reglas, las cuales pueden ser Drop, Reject, Sdrop.

4.9. Módulo Preprocesador

Como se vio en la sección Preprocessors, estos módulos están diseñados para analizar el tráfico de la red previo a que el motor de Snort actúe, de esta forma se puede acceder al dato completo con la información de todos los protocolos de red. En esta sección se revisará cómo construir un módulo propio en la version 2.9.7.3 de Snort.

4.9.1. Estructura básica

Los códigos de los preprocesadores se encuentran en la carpeta preprocessors y tienen un prefijo `spp_`, un ejemplo de ello es `spp_example.c` y `spp_example.h`, donde el primero es el código fuente y el segundo son las cabeceras donde se definen las estructuras que se usarán.
El archivo `spp_example.c` debe tener como base la implementación de las siguiente funciones, para lograr un correcto funcionamiento. En particular si la opcion de reload no esta activada al momento de compilar todas las funciones que cominezan con ese prefijo no son necesarias.

1. Estructura de dato para recibir los parámetros de configuración existentes en "snort.conf"
2. Función Setup, que permite agregar el preprocesador a los registros de Snort.
3. Función Init, para inicializar el procesador en Snort.
4. Function Process, donde el núcleo del proceso, lugar donde llegaran lo paquetes para ser analizados.
5. Function Parse, para asignar los valores a la estructura de datos de configuración.
6. Función Reload, la cual permite recargar los valores del archivo de configuración en caso de que snort se reinicie.
7. Función ReloadVerify, para verificar la correcta carga de parámetros de configuración.
8. Function ReloadSwapPolicyFree, elimina los datos asociados a la estructura de datos con los parámetros de configuración del preprocesador.
9. Funcion ReloadSwap, cambia nuevo tSfPolicyUserContextId por el antiguo.
10. Function ReloadSwapFree, borra los punteros asociados a la estructura de datos con los parametros de configuracion del preprocesador.

Un ejemplo de cómo debería ser una estructura básica para un preprocesador llamado Example es:

```
void SetupExample(void);
static void ExampleInit(struct _SnortConfig *, char *);
static void ExampleProcess(void *, void *);
static ExampleConfig * ExampleParse(char *);
#ifdef SNORT_RELOAD
static void ExampleReload(struct _SnortConfig *, char *, void **);
static int ExampleReloadVerify(struct _SnortConfig *, void *);
static int ExampleReloadSwapPolicyFree(tSfPolicyUserContextId, tSfPolicyId, void *);
static void * ExampleReloadSwap(struct _SnortConfig *, void *);
```

```
static void ExampleReloadSwapFree(void *);
#endif
```

También cada preprocesador puede implementar sus propias funciones, para lograr el su objetivo propio.

Para lograr compilar un nuevo preprocesador hay que agregar este al archivo Makefile.in que esta en la carpeta de preprocessors el nombre del preprocesador en las siguientes lineas

```
am__libspp_a_SOURCES_DIST = spp_example.c spp_example.h
@BUILD_PROCPIDSTATS_TRUE@am__objects_1 = spp_example.$(OBJEXT)
$(PROCPIDSTATS_SOURCE)  spp_example.c spp_example.h
```

Junto con esto agregar la cabecera `spp\_example.h` al archivo `plugbase.c` y agregar a la función `RegisterPreprocessors(void)` la función `SetupExample()`.

4.9.2. Parámetros de configuración

Los preprocesadores obtienen sus parámetros de configuración para la ejecución desde el archivo "snort.conf", esto nos permite cambiar las formas de uso de nuestro preprocesador después de ser compilado. En snort.conf en la sección 5 se encuentran los preprocesadores y es ahí donde se agrega el nombre del preprocesador para que entre en ejecución, un ejemplo de esto se muestra en el siguiente fragmente del archivo de configuración

```
# Back Orifice detection.
preprocessor bo:  noalert { snort_attack }
```

de esta manera el preprocesador "bo" está activo (línea no comentada) y además se le pasa un parámetro de configuración.

Por el lado del código del preprocesador, para poder acceder a estos parámetros de configuración es necesario hacer algunos cambios respecto a las versiones anteriores de Snort. En esta versión del software es necesario definir una nueva forma para obtener los parámetros de configuración desde el archivo de configuración "snort.conf". Esta parte es crucial para poder fijar los parámetros que se usarán para el funcionamiento del preprocesador, dado que una vez compilado el software es la única manera de manipular las funcionalidades de este.
Para esto se implementó una biblioteca llamada `sfPolicyData`, de la familia "sfutil" alojada en la carpeta **sfutil/** la cual permite obtener o fijar los archivos de configuración. Esta biblioteca usa como base las funciones declaradas en `sfPolicy.h` y `sfPolicy.c` las cuales describen la creación y la modificación de las estructuras del tipo Policy.

`sfPolicy` tiene estructuras y funciones para la gestión de los parámetros de configuración, los cuales son presentados a continuación.

Estructuras

- tSfPolicyId: estructura que cuenta con una sola variable `unsigned int` que permite identificar el archivo de configuración de que se está usando.
- tSfPolicy: cuenta con 3 variables, `unsigned int refCount`, `char *filename` y `unsigned int ifConfig Processed`
- tSfPolicyConfig: tiene `tSfPolicy **ppPolicies`, `tSfPolicyId defaultPolicyId`, `tSfPolicyId numAllocatedPolicies` y `unsigned int numActivePolicies`, lo cual permite manipular mas de un archivo de configuración en el caso en el que Snort se quiera correr con parámetros diferentes para VLAN's diferentes.

Funciones

- tSfPolicyConfig * sfPolicyInit(void), inicializa un puntero a una estructura `tSfPolicyConfig`.
- void sfPolicyFini(tSfPolicyConfig *), borra el puntero y sus punteros anidados.
- int sfPolicyAdd(tSfPolicyConfig *,char *), agrega un archivo de configuración a la estructura `tSfPolicyConfig`.
- void sfPolicyDelete(tSfPolicyConfig *, tSfPolicyId), borra los parámetros de configuración asociados a un `tSfPolicyId` determinado de la estructura `tSfPolicyConfig`.
- char * sfPolicyGet(tSfPolicyConfig *, tSfPolicyId), retorna el nombre del archivo de configuración asociado a un `tSfPolicyId`.
- tSfPolicyId sfGetDefaultPolicy(tSfPolicyConfig *config), retorna el `tSfPolicyId` por defecto.
- void sfSetDefaultPolicy(tSfPolicyConfig *config, tSfPolicyId policyId);, fija un `tSfPolicyId` para ser esta la que se use por defecto.
- tSfPolicyId sfPolicyNumAllocated(tSfPolicyConfig *config); retorna el número de archivos de configuración guardados.

más otras funciones que para este estudio no fueron usada.

Para el caso de `sf_Policy_Data.h`, define 2 variables externas del tipo `tSfPolicyId`

- napRuntimePolicyId
- ipsRuntimePolicyId

y funciones para manipular estas 2 variables, las cuales difieren en el modo en el cual se ejecutará Snort, en nuestro caso solo usaremos la napRuntimePolicyId porque estamos ejecutando el software solo para mirar en la red. Alguna de las funciones implementadas son:

- void setNapRuntimePolicy(tSfPolicyId id), fija un valor a `napRuntimePolicyId`.
- tSfPolicyId getNapRuntimePolicy(void), obtiene el valor de `napRuntimePolicyId`.
- int isNapRuntimePolicyDefault(void), revisa si `napRuntimePolicyId` es el `tSfPolicyId` por defecto.
- tSfPolicyId getParserPolicy(SnortConfig *sc), obtiene el `tSfPolicyId` de "sc", si es vacío el campo de "sc" retorna el valor por defecto de `snort\_config`.
- void setParserPolicy(SnortConfig *sc, tSfPolicyId id), fija un `tSfPolicyId` a $sc \rightarrow parserPolicyId$.
- int isParserPolicyDefault(SnortConfig *sc), verifica si $sc \rightarrow parserPolicyId$ o $sc \rightarrow parserPolicyId$ es el valor por defecto.

Para facilitar la creación de instancias de políticas de datos específicos se crean contextos, así el usuario puede crear/borrar contextos, fijar/limpiar/obtener datos de usuario para una política especifica, la por defecto o la actual en uso. El usuario también puede iterar por todas las instancias de datos del usuario. Estas se materializan en el modulo `sfPolicyUserData.c` y `sfPolicyUserData.h` en la cual se define una estructura llamada "tSfPolicyUserContext" que contiene 4 variables, que se describen a continuación.

```
tSfPolicyId  currentPolicyId;
unsigned int numAllocatedPolicies;
unsigned int numActivePolicies;
void **userConfig;
```

Alguna de las funciones implementadas para el manejo de contextos son:

- tSfPolicyUserContextId sfPolicyConfigCreate(void), crea un nuevo contexto de usuario.
- void sfPolicyConfigDelete(tSfPolicyUserContextId pContext), borra un contexto de usuario.
- void * sfPolicyUserDataGet (tSfPolicyUserContextId pContext, tSfPolicyId policyId), retorna el puntero con él la estructura de dato de configuración para esos parametros especificos.

- int sfPolicyUserDataSet (tSfPolicyUserContextId pContext, tSfPolicyId policyId, void *config), fija a un contexto de usuario un puntero a una estructura de datos de un preprocesador a un `tSfPolicyId` específico.
- int sfPolicyUserDataSetCurrent (tSfPolicyUserContextId pContext, void *config), fija a un contexto de usuario un puntero a una estructura de datos de un preprocesador al `tSfPolicyId` actual del contexto de usuario.
- void * sfPolicyUserDataGetCurrent (tSfPolicyUserContextId pContext), retorna el puntero con la estructura de dato de configuración para el `tSfPolicyId` actual en el contexto.
- void *sfPolicyUserDataClear(tSfPolicyUserContextId pContext, tSfPolicyId policyId), borra todos los datos asociados a un `tSfPolicyId` especifico en el contexto de usuario.

Capítulo 5

ANOMALY DETECTION

A diferencia de la técnica de detección de intrusos por firmas, las técnicas de detección por anomalías tienen la cualidad de estar pensadas para la búsqueda de vulneraciones a la seguridad a partir de los datos propios de la red.
Varios fueron los métodos que se mostraron en las secciones anteriores, existiendo una variedad de formas y métodos para alcanzar este objetivo, en esta trabajo se eligió un camino que permitiera la detección en tiempo real y que no requiera de datos de entrenamiento debido al alto costo que tiene generar patrones de comportamientos de la red libre de ataques.

Es así como se se elige la técnica propuesta por Pedro Casas [1] "Unsupervised Network Intrusion Detection System" capaz de detectar ataques de red sin basarse en firmas, entrenamiento o etiquetado de tráfico de ningún tipo. Basándose en la observación de del tráfico de la red en búsqueda de ataques a la red incluso aquellas que tienen poca influencia en el tráfico normal de la red. Esta detección no supervisada busca identificar "valores atípicos"dentro del total de la paquetes capturados en una ventana de tiempo, para luego aplicar una robusta técnica de clasificación para el tráfico que componen un posible ataque.

5.1. Unsupervised Network Intrusion Detection System

El UNIDS está compuesto de 3 pasos consecutivos, analizando el tráfico de la red capturado en una ventana de tiempo (Multi-Resolution Flow Aggregation), hacer una clasificación (Clustering) usando los datos de una ventana sospechosa y luego usar un sistema de umbrales dinámicos para hacer la elección de los ataques. La figura 5.1 muestra las distintas etapas del sistema.

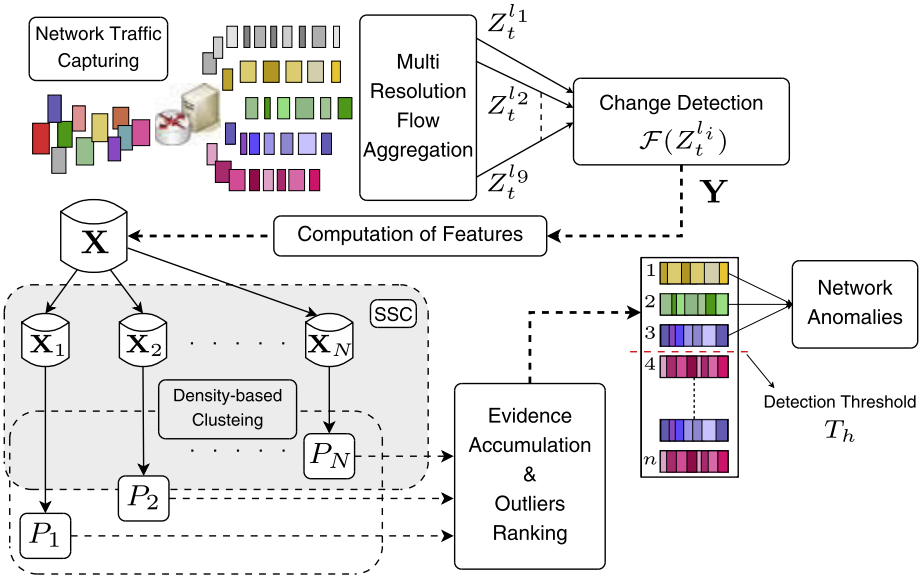


Figura 5.1: Unsupervised Network Intrusion Detection System [1]

5.1.1. Multi-Resolution Flow Aggregation

La primera etapa consiste en detectar de manera general cuando hay una anomalía en una ventana de tiempo fija, la cual será usada a posterior por una etapa de clustering. Para hacer esto, se capturan lo paquetes los cuales son agregados a un flujo de tráfico multi-resolución. Diferentes series de tiempo son construidas sobre este flujo y cualquier método de detección de cambios puede ser usado, en este caso se usa el propuesto por Graham [18] usando como métrica $\#bytes$, $\#packets$ o $\#flows$ por ventana de tiempo. Este algoritmo básicamente marca una anomalía cuando la derivada de uno de estos excede un umbral dinámico basado en el comportamiento global de ese periodo. Esta etapa es crucial en el sistema para reducir la carga de datos que se usará en el clustering, etapa que ciertamente es más costosa en términos de recursos computacionales.

Para lograr reducir la carga de datos que deben ser analizados por el clustering es necesario reducir la cantidad de datos que son capturados. Una opción sería hacer un muestreo del flujo lo cual nos permitiría reducir el número de paquetes pero perdemos resolución de lo que ocurre en la red, Androulidakis [19] habla y desarrolla la idea de hacer muestreos inteligentes que nos permitan reducir el espectro pero considerando todo el tráfico, permitiendo así detectar y clasificar anomalías en la red. Esto está basado en la idea que una gran cantidad de información está contenida en una pequeña fracción del flujo. Si bien no se usará el mismo método la última idea es clave para la implementación de lo que se quiere en esta primera fase.

Encontrar diferencias significativas en los datos de la red

Como se mencionó anteriormente, en esta etapa la idea es crear series de tiempo con distintas resoluciones, detectar las diferencias y que además reducir la dimensión de los datos capturados de una manera inteligente, para ellos recurrimos a técnicas de procesamiento de tráfico de red que nos de un primera alerta de lo que ocurre en ella.

Así es como Carmode propone un algoritmo que reduce la dimensión del tráfico de la red, llevándolo a un pequeño espacio, con tiempos de actualización muy pequeños y garantiza encontrar *Deltoides* significativos con una precisión previamente fijada. La idea de un *Deltoide* hace alusión a un ítem que presenta una gran diferencia, ya sea una diferencia *absoluta*, *relativa* o *variacional*. Este algoritmo fue diseñado para analizar datos de redes de alta velocidad como las usadas por los ISP.

El análisis de este algoritmo usa como entrada solo las cabeceras de los paquetes IP y con ello lograr resúmenes de lo que ocurre en la red en una ventana de tiempo para compararlo con lo que ocurre en la ventana de tiempo anterior, de manera de detectar señales de algo extraño, dado que variaciones en la cantidad de IP fuentes/destino, puertos fuente/destino, bytes, paquetes u otro medida están correlacionadas con ciertos tipos de ataques o escaneos de puertos [20].

Este algoritmo usa *Groups Tests* en una estructura *No-adaptative Combinatorial Groups Testing* que es la base de todo el algoritmo, y tiene la ventaja de no usar ninguna suposición en la entrada.

Antes de seguir con el desarrollo del algoritmo se definirán algunos aspectos importantes para entender de mejor manera el enfoque y la justificación de este.

Anteriormente se mencionó el concepto de *Deltoide* como una gran variación de un ítem, estas podrían ser de diferentes formas:

1. **Diferencias absolutas**: una gran diferencia en el número de paquetes enviados entre una ventana de tiempo y la siguiente.
2. **Diferencias relativas**: una alta tasa de cambio entre el número de paquetes de una ventana de tiempo y la siguiente.
3. **Diferencias variacionales**: una gran variación del número de paquetes tomados sobre múltiples periodos de tiempo.

Cada una de estas formas de ver diferencias sirve para distintas situaciones, por ejemplo cuando un periódicos saca una noticia nueva este recibe muchas solicitudes de manera

abrupta destacando la diferencia absoluta, mientras que un ataque *Flash crowds* podría verse reflejado en las diferencias relativas y diferencias variacionales podrían verse reflejadas en tráficos como los de una oficina donde hay tráfico alto durante el día y bajo por las noches.

Para ser más precisos en el término "gran", tenemos que considerar que este depende del volumen del tráfico y si estamos contando *#paquetes*, *#bytes* o *#flujos*. Teniendo esto presente y considerando que estaremos mirando siempre el mismo link pero en diferentes ventanas de tiempo se considera lo que los administradores de red consideran como anómalo y se basado en un porcentaje de la suma de las diferencias de todos los ítems, el cual puede variar entre el $1\,\% - 5\,\%$. Así esta fracción llamada ϕ nos permitirá diferenciar aquellos que son deltoides de los que no son son deltoides.

Con estos conceptos ya descritos, se procede a detallar en el escenario que se desea enfrentar para definir el modelo matemático de este algoritmo.

Escenario

Los datos recogidos en ventanas de tiempo de interés son representada en flujos $S_1, S_2, \ldots, S_m$, esto puede ser representado como vectores donde S_j representa todos los datos capturados en el tiempo j, cada uno de estos vectores contienen la cantidad de elementos i al final de esa ventana de tiempo. Así $S_j[i]$ representa la cantidad de veces que estuvo presente el ítem i en el tiempo j. La dimensión de S_j es n lo que implica $i \in \{0 \ldots n-1\}$.

En este caso particular se desea capturar paquetes IP por lo que $n = 2^{32}$ y $S_j[i]$ representa el total de flujo para una IP particular en una ventana de tiempo particular. El Flujo de paquetes IP puede ser modelado por el *modelo de caja registradora*, lo que significa que dentro de una misma ventana de tiempo este ítem puede aparecer varias veces, lo que implica que cada una de estas contribuciones agrega un valor a $S_j[i]$. De esta manera como cada paquete tiene una dirección IP i y un tamaño de paquete p tal que $S_j[i] \leftarrow S_j[i] + p$. De esta manera se logra tener tener una representación precisa de lo que ocurre en la red reduciendo la cantidad de información almacenada en $S_j[i]$ y todo esto en tiempo real. Esta forma nos permite hacer consultas del tipo (j, k) para encontrar las diferencias de los ítems i entre S_j y S_k.

Teniendo esta notación podemos redefinir los conceptos de diferencias:

- *Diferencia Absoluta*: la diferencia absoluta para un ítem i es $|S_j[i] - S_k[i]|$.

- *Diferencia relativa*: la diferencia relativa de un ítem i es $S_j[i]/max\{S_k[i], 1\}$[1].
- *Diferencias variacionales*: la diferencia variacional de un ítem i sobre un flujo l se obtiene de $\sum_{j=1}^{l}(S_j[i] - \sum_{k=1}^{l} S_k[i]/l)^2$

De esta manera un ítem que tiene un gran diferencia en cualquiera de las 3 formas pasa a ser un *Deltoide*.

Definición 1 (Deltoide exacto): para cualquier ítem i, fijamos $D[i]$ denota la diferencia de este ítem para un diferencia absoluta, relativa o variacional. Un *ϕ–deltoide* es un item i tal que $D[i] > \phi \sum_x D[x]$.

Para este algoritmo se toma una ligera relajación de este problema, calculando una aproximación del *Deltoide*.

Definición 2 (Deltoide aproximado): tomando $\epsilon \leq \phi$, la ϵ–aproximación de un *ϕ–deltoide* es el problema de encontrar todos los ítems i cuya diferencia $D[i]$ satisface el criterio $D[i] > (\phi + \epsilon) \sum_x D[x]$ y no reporta el ítem cuando $D[i] < (\phi - \epsilon) \sum_x D[x]$. Los ítem entre los umbrales no se tiene certeza si es o no un *Deltoide*

$$i \in Deltoide \Rightarrow D[i] > (\phi + \epsilon) \sum_x D[x]$$
$$i \notin Deltoide \Rightarrow D[i] < (\phi - \epsilon) \sum_x D[x]$$

Este algoritmo tiene una base probabilística con parámetros definidos por el usuario, así δ es la cota superior para la probabilidad de que el algoritmo falle. De esta manera los parámetros que fijan el comportamiento del algoritmo son ϕ, ϵ y δ además de n.

Combinatiorial Gruop Testing

Como se mencionó al comienzo de esta sección uno de los pilares base para el desarrollo de este algoritmo es el famoso y ampliamente usado *Gruop Testing (GT)*.

Un GT considera una población V de n ítems con un pequeño pero desconocido subconjunto defectuosos $D \subseteq V$. El problema está en identificar el conjunto D por una secuencia de *Group Tests*. Cada test tiene 2 posibles resultados en el subconjunto X de V, un resultado puro indica que $X \cap D = 0$, y un resultado contaminado indica que

[1]El 1 es por si $S_k[i] = 0$

$X \cap D \neq 0$. El objetivo es minimizar el número de test bajo el peor escenario.

Los algoritmos GT pueden ser divididos aproximadamente en 2 categorías: *Combinatorial Group Testing (CGT)* o *Probabilistic Group Testing (PGT)*. En CGT es frecuente asumir que el número de defectos para una cantidad n de ítems es igual o a lo sumo d para algún entero positivo fijo. En PGT se fija alguna probabilidad p para tener algún defecto. Las estrategias de GT pueden ser adaptativas o no-adaptativa, un algoritmo GT no-adaptativo se da cuando todos los test deben ser especificados sin conocer el resultado de otros test, esto permite paralelizar esta etapa, por el otro lado los adaptativos toman valores anteriores de otros subconjuntos para hacer el test.

Acá la idea es usar un CGT no-adaptativo diseñando un número el test. En general el procedimiento es el siguiente: por cada inserción de un ítem i a un grupo se determina cuales subconjuntos son incluidos. Cada subconjunto está asociado con un contador y por cada inserción se incrementa el este si cumple con un criterio válido. El test para la identificación de un candidato se basa en verificar si el contador para un subconjunto excede un cierto umbral, bajo ciertas restricciones esto nos dirá si este ítem es un *deltoide* del conjunto. En general el test podría errar con alguna probabilidad, por lo que se necesita una cota para los falsos positivos (ítem que no son deltoides en la salida) y los falsos negativos (falla en incluir un deltoide en la salida).

De esta manera el *no-adaptativo Group Testing* se divide en 2 partes: *identificación*, para encontrar un conjunto de candidatos el que incluye todos los deltoides y una *verificación*, el cual remueve ítems que se fijaron como candidatos sin ser deltoides. Para ambas partes se fija una estructura el cual consiste en un conjunto de "Test" para la estructura de datos, así cuando un ítem llega es agregado a la estructura de datos.

Identificación

Estructura de grupo: Los grupos son subconjuntos de ítems, definidos por una función hash de pares independientes. Tomando un factor de aproximación ϵ y una probabilidad de falla δ, se elige $t_{id} = \log(1/\delta)$ como el número de funciones hash que reducen el dominio de $n \to g$ $h_{1\ldots \log 1/\delta} : \{0 \ldots n-1\} \to \{1 \ldots g\}$. Donde g es el número de grupos que se especificará más adelante.

La creación de las funcione hash de pares independientes están hechas con un método específico llamado funciones hash sobre campo finito p, donde $Z_p = \{0, 1, \ldots, p-1\}$ es el campo con el que se realizan las operaciones de suma y multiplicación. Se fija una semilla aleatoria $s = (a, b) \in Z_p \times Z_p$ construida aleatoria y uniformemente. y se construyen de la siguiente forma:

$$h_s(x) = ((ax) \gg p + b) mod(p)$$

Dado que $(a, x) \in p$ el producto podría exceder el valor de p se realiza un corrimiento a la derecha. En caso particular $p = n$ y el ítem i es llevado a un grupo g con el $mod(g)$ del resultado de $h_a(i)$.

Con lo anterior se fija $G_{a,b} = \{i|h_a(i) = b\}$ para definir que un ítem i al pasar por la función hash a retorne el grupo b asignado.

Tests: dentro de cada grupo se fijan $1 + \log(n)$ estructuras de datos $T_{a,b,c}$. Esto permite plantear un test para cada ítem del grupo. La estructura de datos depende de la naturaleza de las diferencias que estemos intentar detectar la cual se especificará más adelante, por ahora se asume que el test retorna si es un deltoide. Fijamos B_c para referirnos a un conjunto de enteros cuya representación binaria tiene un 1 en la c–ésima posición del bit para $c = 1 \ldots \log n$, por conveniencia de notación fijamos $B_0 = 0 \ldots n - 1$ para llevar la cuenta de todos los ítem que cayeron en ese grupo. Para el resto de las estructuras se debe cumplir $G_{a,b} \cap B_c$ para agregar una cuenta a la estructura, además se asume que la función debe ser lineal de los datos. Se fija además $T'_{a,b,c}$ como el complemento de $T_{a,b,c}$: este último reporta si es un deltoide en $G_{a,b} \cap B_c$, mientras que el complemento reporta si es un deltoide en $G_{a,b}/B_c$. Por linealidad de la función $T'_{a,b,c} = T_{a,b,0} - T_{a,b,c}$. Finalmente para algún test T, se fija $|T|$ para referirse al resultado del test, si $|T| = 1$ significa que el test resultó positivo y $|T| = 0$ para cualquier otro caso.

En la figura 5.2 se muestra como un ítem i es ingresado a las diferentes estructuras de datos dependiendo de la cantidad de funciones hash que se tengan asignadas, cada una de estas asigna el ítem i a un grupo b marcado por la columna amarilla, de este modo se aplica en cada grupo el test $T_{a,b,c}$ que permite saber en qué sub-grupo B_c agregar el valor asociado al ítem i.

Identificación para Group Testing: para encontrar el deltoide entre S_j y S_k necesitamos la combinación de los test de la estructura de dato de cada uno de los stream, $T_{a,b,c}(j)$ y $T_{a,b,c}(k)$ para obtener $T_{a,b,c}(j, k)$. Como esto varía dependiendo del tipo de diferencia que estemos buscando, por ahora solo lo tomaremos como una caja negra la cual retorno si es o no un deltoide. Para ello aplicaremos el siguiente procedimiento:

- Para cada grupo $G_{a,b}$, si $|T_{a,b,0}(j, k)| = 0$ concluimos que no es un deltoide en este grupo y se sigue con el siguiente. Ver caso (c) fig 5.3.
- Para cada valor de c, si $|T_{a,b,c}| = |T'_{a,b,c}|$, implica que ambos son negativos y este no es un deltoide o ambos son positivos lo que implica más de un deltoide en el mismo grupo. En ambos caso se rechaza el grupo. Ver caso (a) fig 5.3.
- Finalmente si $|T_{a,b,c}| = 1$ el test esta correcto y el deltoide $i \in B_c$ por lo que tiene un 1 en el bit c, o en el otro caso si $|T'_{a,b,c}| = 0$ tiene un 0 en el bit c. Así la representación binaria del ítem puede ser recuperada. Ver caso (b) fig 5.3.

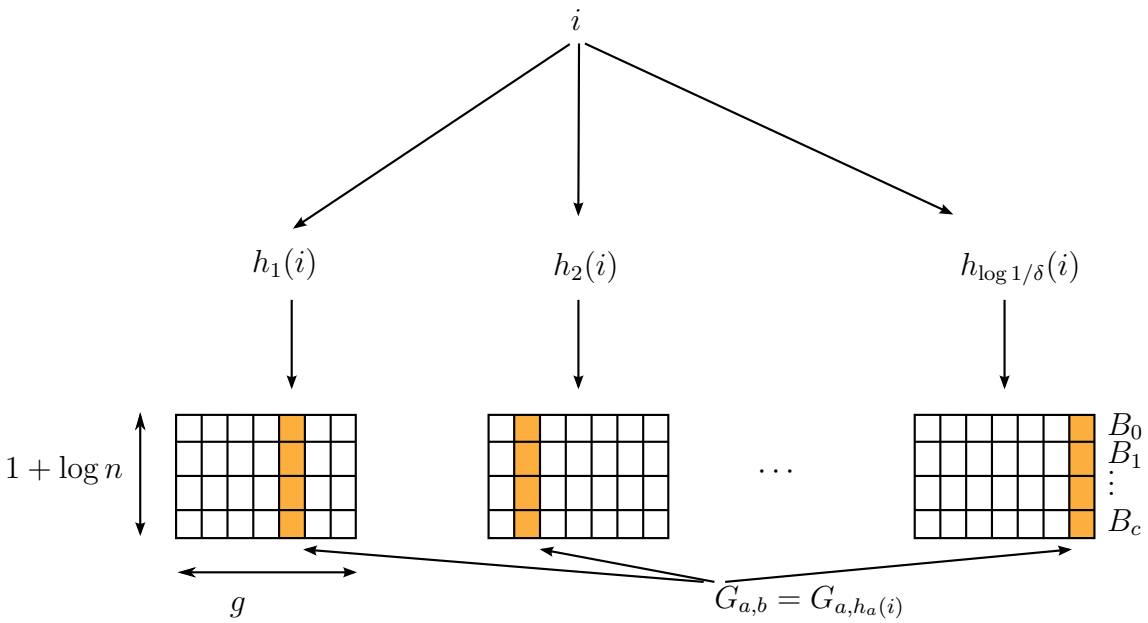


Figura 5.2: Estructura de inserción para un Group Tests

Si el grupo no es rechazado entonces el ítem i es encontrado y es agregado a la lista de los posibles candidatos, el cual se cree pueda ser una ϵ–aproximación de un ϕ–deltoide.

La figura 5.3 muestra los tres casos para el test, en el caso (a) hay 2 ítem en el mismo grupo por lo que al hacer la comprobación $T'_{a,b,3} = T_{a,b,3}$ el ítem se descarta porque se hace imposible recuperar el valor original. En el caso (b) hay un solo ítem mayoritario por lo que es posible recuperar el valor original. En el último caso (c) ningún ítem supera el umbral (marcado en la línea punteada roja) por lo que al hacer el test $T_{a,b,0}$ se descarta inmediatamente sin seguir con el resto de los test. En el caso particular (b) donde es posible recuperar el valor original, la salida del test es la representación binaria del ítem 010100100.

Verificación

En la práctica el test no es perfecto, tiene una probabilidad de falla, el cual puede llevar a falsos positivos. Una verificación simple podría evitar esto. Teniendo en mano los ítem identificados como candidatos ($i \in G_{a,b}$), se realiza una "comprobación de coherencia" verificando que $h_a(i) = b$, si esto no es correcto claramente existió un error y este es rechazado. Para tener una buena garantía de que el ítem encontrado esta correcto se crea una estructura de datos de Verificación. Esta estructura se parece mucho a la de identificación pero con otros parámetros.

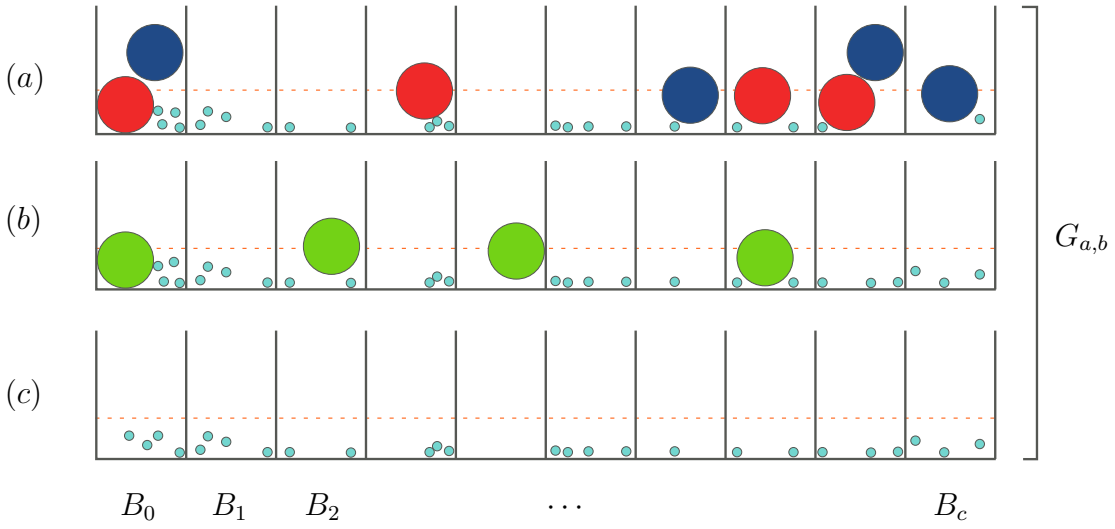


Figura 5.3: Distintos escenarios para el test en los grupos

Grupos y tests: usando los mismos parámetros ϵ y δ se eligen $t_{ver} = 4\log(1/\delta)$ nuevas familias de funciones hash $f_1 \dots f_{4\log(1/\delta)} : \{1 \dots n\} \to \{1 \dots v\}$ generadas por un par de funciones hash independientes. Pero esta vez solo se fija un simple test para la estructura de datos de cada grupo $V_{a,b}$.El test y el número de grupos se elige para que la probabilidad de error de cada test sea a lo más 1/8.

Verificación para Group Testing: para cada ítem candidato, se calcula el grupo al que cae en la estructura de datos de verificación. Para cada uno de estos grupos, se calcula el resultado del test, y tomando la mayoría de los votos de los tests (positivo o negativo) en su totalidad es el resultado del ítem. Si el ítem es positivo entonces tenemos en la salida una ϵ–aproximación de un ϕ–deltoide.

Con esta configuración podemos asegurar que cada ítem que pasa por la identificación y la verificación tienen una probabilidad de a lo más δ de no ser un deltoide. De la misma manera un deltoide tiene una probabilidad de a lo más δ de no pasar por la etapa de verificación.

Procedimiento de actualización El procedimiento completo de actualización para el Combinatorial Gruop Testing es :

Encontrando el Deltoide

La caja negra de la cual se hable anteriormente será descrita a continuación. Además, para cada uno de los tipos de diferencia que deseamos encontrar es necesario fijar un

Algorithm 1 Actualización CGT y VGT

Require: Leer un nuevo ítem i con un tráfico p (bytes, paquetes o flujos)

```
1: function UPDATE(i, p)
2:     for a ← 1 to t_id do
3:         for c ← 0 to log n do
4:             if i ∈ B_c then
5:                 Actualizar T_{a,h_a(i)} con p
6:             end if
7:         end for
8:     end for
9:     for a ← 1tot_verer do
10:        Actualizar V_{a,f_a(i)} con p
11:    end for
12: end function
```

umbral numérico con el cual se definirá si es o no un deltoide $\phi \sum_i D[i]$ el cual también será descrito.

Deltoides Absolutos

Para los Deltoides absolutos, el test para actualizar la estructura de datos es bastante simple y requiere una solo variables, para ello se suman el total de veces que el ítem estuvo presente en la ventana de tiempo por cada test.

$$T_{a,b,c} = \sum_{i \in G_{a,b} \cap B_c} S_j[i]$$

La estructura de datos es lineal y fácil de actualizar, cuando se actualiza un ítem i que llega de tamaño p, se suma p a todos los contadores donde $T_{a,h_a(i),c}$. De la misma manera se define la combinación de test para la ventana de tiempo j y k como

$$T_{a,b,c}(j,k) = |T_{a,b,c}(j) - T_{a,b,c}(k)|$$

$$|T_{a,b,c}(j,k)| = 1 \Leftrightarrow T_{a,b,c}(j,k) > \phi ||S_j - S_k||_1$$

Para fijar la cantidad de grupos en la identificación y en la verificación se utiliza el criterio de $g = 2/\epsilon$ y $v = 8/\epsilon$ como parámetro para este algoritmo.

Fijar el umbral: El umbral en este caso está dado por el cálculo de la norma 1 $||S_j - S_k||_1$, para ello podemos usar una estructura externa o usar las estructuras ya existente, en este caso se reutiliza la misma estructura y al terminar la ventana de tiempo se realiza el cálculo de todas las diferencias usando el valor de B_0.

Así cada ϵ–aproximación de deltoides absolutos es encontrado con este algoritmo con una probabilidad de al menos $1 - \delta$. El espacio en disco para buscar un deltoide es $O((1/\epsilon)\log(n)\log(1/\delta)$. El tiempo que le toma al test actualizar la estructura de dato es $O(\log(n)\log(1/\delta)$ por cada ítem que llega y el costo en tiempo para encontrar un deltoide es $O((1/\epsilon)\log(n)\log(1\delta)$.

Deltoides Variacional

Para encontrar ítems con diferencia variacional alta, primero describimos cómo construir un test y encontrar ítems donde su valor cuadrático es grande, con este cálculo podemos construir un método para encontrar alta varianza en los ítems. Para realizar esto podríamos fijar una nueva estructura de datos pero esto llevaría a más costo de espacio y tiempo, sin embargo se puede fijar un simple contador para cada test, esto es suficiente para encontrar deltoides. Para cada función hash $h_a : \{0 \dots n-1\} \to \{1 \dots d/\epsilon^2\}$ el cual divide los ítem dentro de los grupos (se especifica más adelante), adicionalmente se fija una segunda función hash z_a basada en una familia de funciones hash de cuartetos independiente que mapea los ítem $\{1 \dots n-1\}$ uniformemente sobre $\{+1, -1\}$. Para cada grupo se calcula

$$T_{a,b,c} = \sum_{i \in G_{a,b} \cap B_c} S_j[i] z_a(i)$$

Este test al igual que el anterior es fácil de mantener S_j se presenta como el modelo de una caja registradora, así por cada actualización solo se tienen que agregar el flujo $S_j[i]$ multiplicado por $z_a(i)$ para todos los valores de a y c.

Para cada grupo donde cayeron items i, $T^2_{a,b,c}$ es una buena estimación para $S^2_j[i]$, con una probabilidad de al menos $2d/(d-1)^2$ que $|T^2_{a,b,c} - S_j[i]^2| < \epsilon||S_j||^2$ [18].

De esta manera la contribución a la varianza del ítem i del flujo l está dado por

$$\sigma^2[i] = \sum_{j=1}^{l}(S_j[i] - \mu[i])^2 \quad \mu[i] = \sum_{k=1}^{l} \frac{S_k[i]}{l}$$

Por linealidad de la función test se puede calcular un estimado individual para el j-ésimo término en esta suma.

$$\sigma^2(j)[i] = (T_{a,b,c}(j) - \sum_{k=1}^{l} T_{a,b,c}(k)/l)^2$$

Para referirnos al total de las varianzas de todos los items $\sum_i \sigma^2[i]$ se usará $\sigma^2(l)$.

Los parámetros usados para la construcción de las estructuras de datos están fijadas a $d = 6$, $g = 6/\epsilon$, $v = 18\epsilon^2$ y el test para identificar deltoides variacionales está dado por

$$T_{a,b,c}(l) = \sum_{j=1}^{l} \sigma^2(j)[i]$$

$$|T_{a,b,c}(l)| = 1 \Leftrightarrow T_{a,b,c}(l) > \phi\sigma^2(l)$$

Cálculo del umbral: para obtener $\phi\sigma^2$ es necesario tener el cálculo de σ^2, para esto podríamos usar una nueva estructura de datos pero estaríamos generando la misma que se usa para la verificación.

$$media_a \sum_b \left(\sum_{j=1}^{l} \left(V_{a,b}(j) - \sum_{k=1}^{l} \frac{V_{a,b}(k)}{l} \right) \right)$$

De esta manera cada ϵ–aproximación de un deltoide variacional es encontrado por el algoritmo con una probabilidad al menos $1 - \delta$. El espacio usado para almacenar la estructura de datos es $O((1/\epsilon^2)\log(n)\log(1/\delta)$. El tiempo que toma una actualización a la estructura de datos es $O(\log(n)\log(1/\delta))$ y el costo en tiempo para encontrar los deltoides es $O((1/\epsilon^2)\log(n)l\log(1/\delta)$.

Deltoides relativos

Encontrar deltoides relativos es encontrar grandes tasas de cambios en los flujos, para realizar esto se hace un aproximación debido a que este cálculo requiere de una gran cantidad de espacio para hacerlo con precisión. En lugar de ello se usa una notación ligeramente más débil de la notación clásica para la aproximación del deltoides. Para Ello es necesario transformar uno de los flujos. Este método no trabaja en general con el modelo de la caja registradora, pero sí requiere que uno de los flujos sea agregado, con este método. Esto permite que el test para uno de los flujos sea calculado en tiempo real y el deltoide puede ser encontrado entre este flujo y uno pre-calculado con anterioridad. Fijamos así $S_{1/k}$ para el flujo cuyo i-ésima entidad es $S_{1/k}[i] = 1/S_k[i]$. De esta manera encontrar un ítem i que tenga una gran diferencia relativa significa encontrar un item i tal que $D[i] = S_k * S_{1/k}$ sea grande, todo esto relativo a $\sum_i D[k]$. Por notación se usará $1/k$ para referirse al flujo "invertido" $S_{1/k}$. Para este método se usa $g = 2/\epsilon$ y $v = 4\epsilon$ y además el test para los deltoides relativos se calcula de la siguiente forma:

$$T_{a,b,c}(j) = \sum_{i \in G_{a,b} \cap B_c} S_j[i]$$

$$T_{a,b,c}(1/k) = \sum_{i \in G_{a,b} \cap B_c} S_{1/k}[i]$$

Y los test combinados $T_{a,b,c}(j, 1/k) = T_{a,b,c}(j) * T_{a,b,c}(1/k)$ [18]

$$|T_{a,b,c}\left(j, \frac{1}{k}\right)| = 1 \leftrightarrow T_{a,b,c}(j, 1/k) > \phi \sum_i \frac{S_j[i]}{S_k[i]}$$

Cálculo del umbral: como en los casos anteriores podemos construir una estructura particular para esta tarea y actualizar por cada ítem que llegue, o usar los valores de la estructura de verificación al final de cada flujo o ventana de tiempo.

$$media_a \sum_b \left(\frac{V_{a,b}(j)}{V_{a,b}(k)}\right)$$

Finalmente esta ϵ–aproximación para los deltoides relativos tiene la posibilidad de encontrar un deltoide con una probabilidad de al menos $1 - \delta$. El espacio usado para almacenar la estructura de datos es $O((1/\epsilon)\log(n)\log(1/\delta)$. El tiempo que toma una actualización a la estructura de datos es $O(\log(n)\log(1/\delta))$ y el costo en tiempo para encontrar los deltoides es $O((1/\epsilon)\log(n)\log(1/\delta)$.

Parámetros experimentales

Deltoides Absolutos

Según Carmode, no hay item cuya diferencia supere el 10 % del total de tráfico de la red, muy pocos logran el 5 % y los valores más comunes se encuentran entre el 1 % y 0,5 %. de esta manera se fija $\phi = 0{,}1\,\%$ para obtener una cantidad considerable de deltoides para luego ajustar al valor acorde a la red. Valores óptimos para este método de detección pueden ser fijados a $\epsilon = \phi/10$ y $\delta = 0{,}25$ es suficiente para dar garantías de detección.

Deltoides Variacionales

El rendimiento de esta modalidad es bastante buena y tiene buenos resultados cuando $\epsilon > \phi$, esto se debe a que es un indicador más complejo, tiene más información procesada respecto al comportamiento, variación sobre el promedio. Valores óptimos para este modelo de detección se encuentran con $\epsilon = \phi$ y $\delta = 0{,}25$.

Deltoides Relativos

En este caso fijar el valor de ϕ es mas importante, fijar un valor muy bajo hará que todos los ítem sean deltoides y fijar un valor muy grande de este no retornara deltoides a la salida. Así resultados aceptables se pueden encontrar en cuando $\epsilon = \phi/3$ y resultados perfectos con $\epsilon = \phi/10$ y el valor de ϕ debe revisarse en la red particular de manera empírica, además valores bajos como $\delta = 0{,}25$ dan buenos resultados.

Los costos asociados a δ tiene una relación inversa con el tiempo de inserción de un nuevo ítem, mientras más bajo el valor de δ más tiempo tarda la inserción porque son más las funciones hash las que hay que procesar, mientras que ϵ tiene un directa relación con el espacio requerido para la búsqueda y almacenamiento de deltoides y no tiene que ver con los tiempos de actualización.

5.1.2. Clustering

La segunda etapa en este algoritmo es la clasificación de todos los paquetes capturados en la ventana de tiempo donde la etapa anterior detectó algún patrón anomalía en alguno de sus flujos de tráfico. Para lograr esta tarea se emplea una combinación de técnicas de clasificación, Sub-Space Clustering (SSC), Density-based Clustering y Evidence Accumulation Clustering (EAC) para lograr un nivel de precisión y robustez lo suficientemente confiable. El resultado de esta clasificación nos entrega un información que puede ser ordenada por el grado de anormalidad de todos los flujos aislados

construyendo así la entrada para la tercera parte.

Para hacer esto se toma los flujos con menor resolución como IPsrc y IPdst. el Tráfico anormal puede ser agrupado aproximadamente en 2 diferentes clases, dependiendo de su estructura espacial y del número de flujo IP impactado. Anomalías del tipo $1 \rightarrow N$ que involucra muchos flujos IP de la misma fuente a diferentes destinos, un ejemplo de esto son los network scans y spreading worms/virus, por otro lado tenemos anomalías del tipo $N \rightarrow 1$ que involucran muchas flujos IP de diferentes fuentes al mismo destino; ejemplo de esto son los ataques DDOS y lso flash-crowds, el $1 \rightarrow 1$ es un caso particular de este último, mientras el caso $N \rightarrow N$ puede ser analizado como múltiples $1 \rightarrow N$ o $N \rightarrow 1$ ataques.
El uso del flujo IPsrc permite destacar los ataques del tipo $1 \rightarrow N$ mientras que los flujos IPdst permiten destacar los ataques del tipo IPdst.La elección de ambos flujos permite asegurar una siempre tendremos anomalías altamente distribuida, donde cualquiera de ellos al presentar un gran numero de de flujos IP podría representar un valor atípico.

Sin perder generalidad, fijemos $\mathbf{Y} = \{\mathbf{y_1}, \ldots, \mathbf{y_n}\}$ el conjunto de n flujos agregados (ej. IPsrc o IPdst) marcados en la ventana de tiempo. Para cada $\mathbf{y_i} \in \mathbf{Y}$ describe por un conjunto de m atributos o características del tráfico, como el número de fuentes y puertos, o tasa de paquetes.. Fijemos $\mathbf{x_i} \in \mathbb{R}^{\mathbf{m}}$ es el vector de características que describen al flujo $\mathbf{y_i}$, y $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^{\mathbf{n \times m}}\}$ la matriz completa de características, describiendo las características espaciales.

UNIDS se basa en en una técnica de clasificación aplicada a $\mathbf{X}$. El objetivo de la clasificación es hacer particiones a conjuntos de de muestras no etiquetadas dentro de un grupo homogéneo de similares características o clusters, basado en alguna medición de similitud definida para el espacio de muestra. Existen muchas notaciones de de similitud o *distancias* diferentes que son usadas en el dominios de las calificaciones, partiendo de la simple distancia Euclidiana a las más complejas notaciones como una similitud basada en estadísticas. Muestras que no pertenecen a ninguno de los cluster existentes son clasificados como valores atípicos. El principal objetivo de esta etapa es identificar estos valores atípicos que se diferencia notablemente del resto de las muestras y adicionalmente categorizar qué tan diferentes son. El enfoque más apropiado para encontrar estos es hacer clusters e identificar nos que no son parte de estos.

Desafortunadamente existen cientos de algoritmos para hacer clasificaciones lo que hace muy difícil la elección de cuál de ellos elegir, que permita encontrar todos los cluster, de todas las formas y tamaños posibles, en general cada uno hace sus propias particiones de datos y siempre el mismo algoritmos entrega diferentes resultados para el mismo conjunto de datos dependiendo de los valores de inicialización y/o diferentes paráme-

tros del algoritmo. Esto es uno de los principales inconvenientes actualmente para las técnicas de análisis de clasificación: la falta de robustez.

Para evitar este tipo de limitación, se desarrolla un enfoque de clasificación dividir y conquistar, usando la notación de conjunto de clustering y múltiples combinaciones de clustering. Para ello se toma la información que provee múltiples particiones de $\mathbf{X}$. De esta manera un conjunto de clusters P consiste en en múltiples particiones P_i producidos por el mismo dato. Cada partición provee una evidencia independiente de una estructura de dato., la que puede ser combinada para construir una nueva medición de similitud que mejor represente la naturaleza de los grupos y los valores atípicos. Hay diferentes caminos para hacer conjuntos de clustering, usando algoritmos juntos o usando el mismo algoritmo pero con diferentes parámetros o inicializaciones. Para este caso particular se usa Sub-Space Clustering (SSC) [21] para producir múltiples particiones de datos, haciendo clasificación basada en Densidad en N diferentes sub-espacios $\mathbf{X_i}$ del espacio original. Una versión gráfica en fig (5.1).

Para cada uno de los N Sub-espacios $\mathbf{X_i} \in \mathbf{X}$ es obtenido por la proyección de $\mathbf{X}$ dentro de las k características de los m atributos, resultando los N K-dimensiones de los sub-espacios. Para explorar en profundidad las características del espacio, el número de de sub-espacios N que son analizados corresponde al número de las k combinaciones obtenidas a partir de m. En la fig. (5.4) se puede ver como se hace la separación y cada uno de estos sub-espacios de 2 dimensiones es utilizado para hacer clasificaciones por densidad. Cada partición P_i es obtenida aplicando DBSCAN [22] al sub-espacio $\mathbf{X_i}$. DBSCAN es un poderoso algoritmo de clasificación que descubre cluster de forma y tamaños arbitrarios, los cluster son separados por áreas de alta densidad en el espacio de áreas de baja densidad. Este algoritmo se ajusta perfectamente al análisis del tráfico de la red porque no necesita a especificación a priori de parámetros difíciles de ajustar como por ejemplo el número de cluster a identificar.¸

El resultado obtenido al aplicar DBSCAN al sub-espacio $\mathbf{X_i}$ son dos: un conjunto de $p(i)$ clusters $\left\{C_1^i, C_2^i, \cdot, C_{p(i)}^i\right\}$ y un conjunto de $q(i)$ valores atípicos $\left\{o_1^i, o_2^i, \cdot, o_{p(i)}^i\right\}$. Para fijar el número de dimensiones k de cada subespacio, se toma una propiedad muy usada de monotonía en el conjunto de clasificación, conocido como la propiedad "downward-closere", si un conjunto de elementos es un cluster en un espacio k-dimensional, cuando este conjunto es parte de un cluster en cualquier (k-1) proyecciones de ese espacio. Esto implica directamente que si existe evidencia de de densidad en $\mathbf{X}$, probablemente esté presente en sub-espacios de menor dimensión. Usando pequeños valores de k proporciona varias ventajas: primero, hacer clasificaciones en espacios de baja dimensionalidad es más eficiente y más rápido que hacer clasificación en dimensiones mayores. Segundo, Algoritmos para clasificación por densidad como DBSCAN proporcionan mejores resultados en espacios de menor dimensión, porque espacios de alta dimensionalidad son usualmente escasos, haciendo más difícil distinguir regiones de mayor densidad con

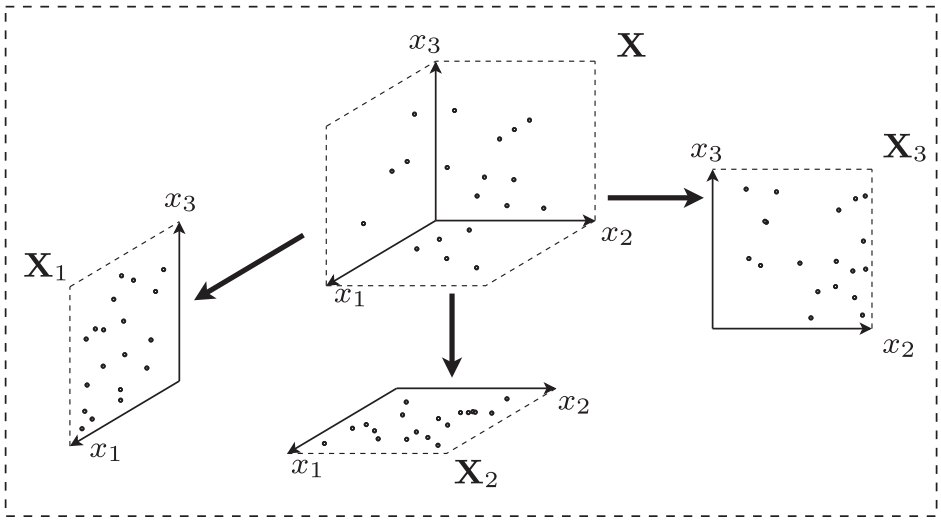


Figura 5.4: Clasificación por Sub-espacio: Los sub-espacios de 2-dimensiones $\mathbf{X_1}$, $\mathbf{X_2}$ y $\mathbf{X_3}$ son obtenidos a partir de un espacio de 3-dimensiones $\mathbf{X}$ por simple proyección

regiones de menor densidad. Finalmente clasificar multimes sub-espacios de baja dimensionalidad proporcionan un análisis con una granularidad más fina, lo cual mejora la capacidad de detectar ataques de muchas características diferentes. Por lo tanto usaremos $k = 2$ para la clasificación de sub-espacios, lo que da $N = m(m-1)/2$ particiones.

Teniendo la información de las N particiones con los cluster y los valores atípicos, es necesario distinguir cuáles de ellos son ataque o anomalías en la red. Para ello se implementa un un tipo de clasificación basado en acumulación de evidencia (EAC), usando múltiples particiones $p(i)$para producir una nueva medición de similitud entre muestras que mejor refleje su agrupación natural.
Un algoritmo particular de EAC es conocido como EA4RO (Evidence Accumulation for Ranking Outliers) el cual realiza una medición de similitud entre los n diferentes flujos agregados descritos en $\mathbf{X}$. EA4RO construye un vector de diferencias $D \in \mathbb{R}^n$ en el cual se acumulan las distancias entre los diferentes valores atípicos o_j^i encontrado en cada sub-espacio $i = 1, \cdot, N$ y los centroides de del mayor cluster del sub-espacio C_{max}^i. La idea detrás de EA4RO es claramente destacar aquellos flujos agregados que son realmente diferentes de la operación normal de tráfico de la red en cada uno de los diferentes sub-espacios, estadísticamente representado por C_{max}^i. Así ataques altamente distribuidos tienen cientos de flujos de IP que pueden ser representados como un valor atípico.

En el algoritmo 2 se presenta un pseudocódigo para EA4RO: Los diferentes parámetros usados por EA4RO son fijados automáticamente por el mismo algoritmo. Los primeros 2 parámetros son usados por el clustering basado en densidad : n_{min} especifica el mínimo número de flujos que pueden ser clasificados por el cluster, mientras que δ_i indica

la máxima distancia de la vecindad de una muestra para identificar regiones densa. n_{min} es fijado por la inicialización del algoritmo, simplemente como una fracción del número total de flujos n a analizar ($\alpha = 5\,\%$ puede ser un buen valor). δ_i es fijado como una fracción del promedio de las distancias entre el flujo los sub-espacios $\mathbf{X_i}$ (un buen valor puede ser 1/10), el cual es estimado del 10 % de los flujos, elegidos aleatoriamente. Esto permite aumentar la velocidad de cómputo. El factor de peso w_i es usado para aumentar los valores atípicos, acá toman más relevancia los valores atípicos que son "menos probables". w_i toma valores altos cuando el tamaño de n_{max_i} del cluster C^i_{max} encierra al número total de flujos n, lo que implica que este valor atípico es particularmente diferente de la mayoría. Finalmente, en lugar de usar la simple distancia Euclidiana como medición de las diferencias, se realiza la distancia de Mahalanobis d_M entre el valor atípico y el centroide del cluster mayor. La distancia de Mahalanobis toma en cuenta la correlación entre muestras, dividiendo la simple distancia Euclidiana por la varianza de las muestras. Esto permite agrandar el grado de anormalidad del valor atípico cuando la varianza es pequeña.

Algorithm 2 Ranking de valores atípicos vía EA4RO

Require: Fijar vector de diferencias D a null de tamaño $n \times 1$
Require: Fijar pequeños cluster de tamaño $n_{min} = \alpha \cdot n$

1: **for** $i \leftarrow 1$ to N **do**
2: Fijar la densidad de la vecindad δ_i para DBSCAN
3: $P_i = DBSCAN(\mathbf{X_i}, \delta_i, n_{min})$
4: Actualizar $D(j)$, $\forall$ valor atípico $o^i_j \in P_i$:
5: $w_i \leftarrow \frac{n}{(n - n_{max_i}) + \epsilon}$
6: $D(j) \leftarrow D(j) + d_m(o^i_j, C^i_{max}) w_i$
7: **end for**
8: Ordenar los flujos: $D_{rank} = sort(D)$
9: Fijar el umbral de detección: $T_h = find\_slope\_break(D_{rank})$

5.1.3. Top-Ranking

Esta última etapa toma la lista ordenada de los flujos aislados del clustering y se aplica un simple umbral de selección de los más importantes. Para ello se realiza el último punto de del algoritmo 1, ordenando el vector D acorde a las distancias de las diferencias y se fija el umbral T_h para la detección de anomalías. Este umbral simplemente se calcula tomando en cuenta aquellos valores de mayor variación.

Capítulo 6

IMPLEMENTACIÓN

Consideraciones de la implementación

En la implementación real de este trabajo de título sólo se avanzo en la primera etapa de la detección de anomalías descrita en la sección 5 para la versión 2,9,7,3 de SNORT, considerando solo las diferencias absolutas. Para la detección de anomalías además de las series de tiempo para las IP fuente/destino se incorporan varias series de tiempo más, que agregan puerto fuente/destino y la cantidad de Bytes asociado a cada uno de estos flujos. Quedando IP_{src}, IP_{dst}, $IP_{src}PORTsrc$, $IP_{dst}PORTdst$, $IP_{src/dst}PORTsrc$, $IP_{src/dst}PORTdst$, $IP_{src/dst}PORTsrc/dst$ los cuales representan en cada caso el ítem i que se procesa.

Para mejorar la visualización de las series de tiempo y ver lo que ocurre en la red para luego ver como esto afectará a las siguientes etapas, corroborar la correcta configuración de parámetros ϕ, δ y ϵ y visualizar la salida de SNORT se implementó el sistema ELK (ElasticSearch, Logstash y Kibana) que es un conjunto de software que permitan recolectar, almacenar y visualizar cualquier tipo de de archivo de texto plano.

El código de la implementación se encuentra completo en GitHub (https://github.com/creyesp/AnomalyDetection) y en esta sección se comentaran las principales componente de funcionamiento, además se describirá las especificaciones del sistema de visualizacion..

6.1. Módulo Anomaly Detection como preprocesador de SNORT

Como se mencionó en la sección de construcción de módulos preprocesadores en SNORT 4.9, se requiere un estructura básica la cual es requisito para el funcionamiento y reconocimiento de SNORT. Las funciones *Reload** no se implementaron, lo cual no afecta el funcionamiento básico, estas están diseñadas para el caso en que SNORT se reinicia

como servicio. De esta manera las funciones principales que se implementan son:

```
void SetupAnomalyDetection(void)
void AnomalyDetectionInit(struct _SnortConfig *sc, char *args)
void ParseAnomalyDetectionArgs(AnomalydetectionConfig* pc, char *args)
void PreprocFunction(Packet *p,void *context)
```

La primera función `SetupAnomalyDetection()` informa a SNORT del preprocesador, asignando el nombre del preprocesador *"AnomalyDetection"* y la función con la que se inicializa, en este caso `AnomalyDetectionInit()`.

En el caso de `AnomalyDetectionInit()` tiene como entrada un puntero a una estructura con los parametros de configuracion de "snort.conf" asociados a este preprocesador y un puntero a argumentos de tipo string. Esta función permite pasar a la estructura central de SNORT las funciones vinculadas al funcionamiento general, como la función que se debe llamar cuando llega un nuevo paquete, se cierra el programa, el informe de rendimiento entregado al final de la ejecución o por ej. el mensaje cuando se está cargando el preprocesar al inicio. Además se llama a la función para asignar los parámetros del archivo de configuración a las variables propias del proprocesador `ParseAnomalyDetectionArgs()`. Por último se inicializan todas las estructuras de CGT y VGT, además de abrir los archivos donde se registran los deltoides.

Y la función central del procesador es `PreprocFunction()` el cual tiene como entrada cada uno de los paquetes que captura SNORT, los que son agregados a las estructuras CGT y VGT respectivamente. Al completarse la ventana de tiempo se revisan las estructuras de datos en busqueda de los Deltoides respecto a la ventana de tiempo anterior. Además en esta función se exportan los Deltoides a un archivo de registro, que deja la hora en la que ocurrió, la IP_{src}, IP_{dst}, $PORT_{src}$ o $PORT_{dst}$ además de la cantidad de paquetes y bytes asociados a cada Deltoide.

Las funcione auxiliares más importantes que complementan el funcionamiento de preprocesador son:

```
static void addGT(Packet *p)
static int ComputeDiffThresh(const CGT_type *cgt)
void writeOutput( FILE* outputfile, unsigned int ** outputList , char tsName[])
```

La función `addGT()` toma el paquete que llega y obtiene la información de la cabecera de cada uno de los paquetes, en este caso solo los IP, y agregando dependiendo de la serie de tiempo que se desee registrar los datos IP_{src}, IP_{dst}, $PORT_{src}$, $PORT_{dst}$, $\#Packets$ y $\#Bytes$ del payload. Además de manera transitoria en esta versión se registra todo la informacion del tráfico anteriormente descrita, junto con el protocolo y si es un paquete TCP de registra también el flag de la cabecera.

Tomando los datos de las estructuras VGT se calcula el umbral con la función `ComputeDiffThresh()` el cual calcula la diferencia de cada uno de los grupos entre las 2 ventanas de tiempo, al final elige la media de ellas para multiplicarda por ϕ, fijando así el umbral de detección.

Las última función secundaria es `writeOutput()`, la cual toma como entrada la lista de Deltoides encontrados en la ventana de tiempo, el puntero donde se van a escribir los archivos y el nombre de los deltoides que se estan escribiendo para las salida por consola. La lista de deltoides contiene en su primer elemento la cantidad de deltoudes de la lista y la cantidad de elementos asociados a cada deltoide, las posibles combinaciones permite escribir IP_{src} o IP_{dst}, $IP_{src/dst}$, $IP_{src/dst}$–$PORT_{src/dst}$, $IP_{src/dst}$–$PORT_{src}$ o $IP_{src/dst}$–$PORT_{dst}$. Además son estas las que opcionalmente permiten visualizar por pantalla la salida de los deltoides.

Por último existe una biblioteca extra para creación, actualización y eliminación de los *Groups Test* llamada cgt.c y cgt.h, basada en un trabajo de Carmode y actualizada a este contexto, la cual tiene alguna de las siguientes funciones:

```
void CGT_Init(CGT_type **, int, int, int);
void CGT_Update( CGT_type *, unsigned int, int,int);
void CGT_Update64( CGT_type *, unsigned int,unsigned int, int,int);
void CGT_Update96( CGT_type *, unsigned int,unsigned int,
                unsigned short int,unsigned short int, int,int);
int CGT_Output( unsigned int ***, CGT_type *, VGT_type *, int);
int CGT_Output64( unsigned int ***, CGT_type *, VGT_type *, int);
int CGT_Output96( unsigned int ***, CGT_type *, VGT_type *, int);
void CGT_Destroy(CGT_type *);

void VGT_Init(VGT_type **,int, int);
void VGT_Update( VGT_type *, unsigned int, int);
void VGT_Update64( VGT_type *, unsigned int,unsigned int, int);
void VGT_Update96( VGT_type *, unsigned int,unsigned int,
                unsigned short int,unsigned short int, int);
void VGT_Destroy(VGT_type *);

unsigned int testCGT(unsigned int rtest[1], const int *countCGT, int thresh)
unsigned int testCGT64(unsigned int rtest[2], const int *countCGT, int thresh)
unsigned int testCGT96(unsigned int rtest[3], const int *countCGT, int thresh)

void loginsert( int *lists, unsigned int val, int diff, int dsize)
void loginsert64( int *lists, unsigned int val1, unsigned int val2,
                int diff, int dsize)
void loginsert96( int *lists, unsigned int val1, unsigned int val2,
                unsigned int val3, int diff, int dsize)
```

Esta biblioteca maneja 2 estructuras de datos, una para los CGT y otra para los VGT, cada una de ellas tiene el número de test que se realizan, las semillas de las funciones hash, la cantidad de grupos y subgrupos, esta ultima solo para el caso CGT. Además de un contador global de la cantidad de paquetes que se están agregando a los grupos. a continuación se muestra las estructuras de datos.

```
typedef struct CGT_type{
```

```
int tests;
int logn;
int buckets;
int subbuckets;
int count;
int ** counts;
long  *testa, *testb;
} CGT_type;

typedef struct VGT_type{
int tests;
int buckets;
int count;
int * counts;
long   *testa, *testb;
} VGT_type;
```

Las estructuras son inicializadas con la función `CGT_Init()` y `VGT_Init()` tomando como parámetros los valores que existen en el archivo de configuración. Así en el caso de CGT $counts = test \times buckets$ punteros a *subuckets* variables de tipo long long donde se almacenan los contados por cada bit. Para el caso de VGT solo se necesitan $counts = tests \times buckets$ variables del tipo long long para almacenar la cuenta de cada ítem.

Para actualizar las estructuras CGT y VGT se utilizan las funciones `CGT_Update()`, `CGT_Update64()` y `CGT_Update96()` dependiendo del tamaño de subgrupos que se esté utilizando, en términos generales lo que se hace en cada una de estas funciones es tomar el ítem i (que puede ser cualquiera de las mencionadas anteriormente) pasarlo por las funciones hash, y el valor retornado se le calcula el módulo del tamaño de grupos y este valor es el grupo asignado para ese ítem, posteriormente se calcula la suma de la cantidad asociada al ítem i. esto se realiza con la función `loginsert()`, `loginsert64()` y `loginsert96()` que recorre cada uno de los subgrupos revisando si el el bit c del ítem es 1, agrega el valor a este test del grupo para ver en qué subgrupos hay que subgrupo. Para el caso de VGT es el mismo procedimiento solo que no es necesario usar las funciones `loginsert()` porque solo tiene un contador.

La búsqueda de deltoides se realiza al final de la ventana de tiempo en la función `PreprocFunction()` con la funciones `CGT_Output()`, `CGT_Output64()` o `CGT_Output96()` dependiendo del largo del item. La función `CGT_Output` recorre la estructura CGT realizando las pruebas descritas en la sección 5.1.1, usando las funciones `testCGT()`, `testCGT64()` o `testCGT96()`, las cuales al resultar exitosas retornan el ítem i posible deltoide que luego es verificado por la estructura VGT. Si todo resulta correcto, todos los candidatos son ordenados y los repetidos son eliminados, generando así la una lista con la cantidad de Deltoides encontrados.

Por último para fijar los parámetros de funcionamiento de SNORT hay que modificar el archivo “snort.conf” y para el caso particular del preprocesador de detección de

alert	permite generar alarmas.
log	activa el registro de deltoides.
time	periodo de tiempo en el que se recolectarán los datos para ser procesados (en segundos).
LogPath	ruta donde se registraron las series de tiempo.
verbose	para habilitar la la salida de los deltoides por consola.
dataflow	activa el registro de todas las cabeceras de los paquetes procesados.
phi	porcentaje del total de tráfico que se usará como umbral de deltoide.
epsilon	porcentaje de aproximación para encontrar un deltoide.
delta	probabilidad de falla del algoritmo.

Tabla 6.1: Parámetros AD en *snort.conf*

anomalías los parámetros configurables son los mostrados en la tabla 6.1

Estos parámetros deben ser puestos en la sección de preprocesadores en el archivo de configuración de la siguiente forma.

```
preprocessor AnomalyDetection: LogPath /var/log/snort/ \
alert \
log \
verbose \
dataflow \
time 3 \
phi 0.01 \
epsilon 0.0008 \
delta 0.063 \
```

6.2. ELK - ElasticSearch Logstash Kibana

Como se mencionó al comienzo de este capítulo para la visualización de del las salidas de SNORT se busca usar un sistema que sea capaz de procesar y visualizar grandes cantidades de datos y que las búsquedas de notificaciones sea fácil de manejar. Para realizar esto existe un conjunto de herramientas llamado ELK, por las iniciales de los 3 software que usa, Elasticsearch, Logstash y Kibana, el cual permite crear un interactivo dashboard a partir de datos en bruto.

6.2.1. Logstash

Logstash es un colector de datos de código abierto capaz de procesar los datos en tiempo real. Logstash puede unificar diferentes fuentes, procesar los datos y enviarlos a un

Figura 6.1: Logos del conjunto ELK

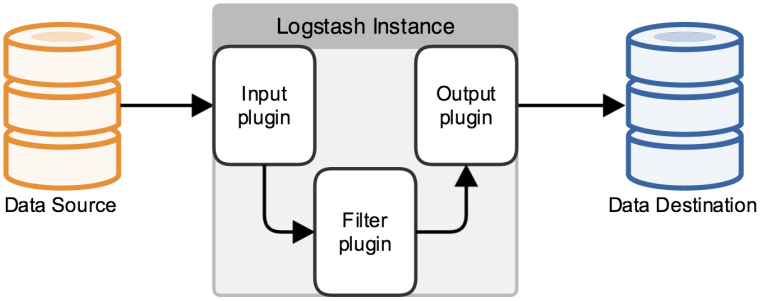


Figura 6.2: Estructura de funcionamiento de Logstash

destino elegido previamente y ser usado por otras aplicaciones de manera centralizada y normalizada, siendo así una herramienta muy útil al momento de procesar datos como los usado por SNORT.

Originalmente Logstash fue diseñado como un colector de log, pero hoy su desarrollo permite procesar casi cualquier tipo de datos gracias a un conjunto de plugins de entrada. Estos datos pueden ser filtrados y transformados por diferentes filtros predefinidos para cierto tipo de archivo, ej log de Apache o crear uno propio que permita darle coherencia a los datos leídos, esto permite separarlos en campos diferentes para así ser procesados a posterior con gran facilidad. Y por último Logstash tienen un conjunto de plugins de salida que permiten conectar la salida de este con la entrada de otros programas o formatos de archivo, en este caso nos interesa la combinación Logstash-ElasticSearch. La figura 6.2 muestra modularmente el funcionamiento de Logstash.

Logstash es fácil de usar y de configurar, por cada tipo archivo que deseamos recolectar se debe crear un archivo de configuración con la siguiente estructura, *input*, *filter* y *output*. Como ejemplo se muestra el archivo de configuración usado para leer el flujo completo de datos usados por el preprocesador, donde se convierten los campos numéricos a flotante y en el caso de las IP y los puertos se dejan como string. Estos son luego enviarlos a elasticsearch y al standard output.

```
input {
file {
path => "/var/log/snort/IPsdPORTsd.csv"
}
```

```
}
filter {
csv {
separator => ","
columns => ["DATE","IP_SRC","IP_DST","PORT_SRC","PORT_DST","PACKETS","xDSIZE"]

}
mutate {convert => ["xDSIZE", "float"]}
mutate {convert => ["PACKETS", "float"]}
mutate {convert => ["PORT_SRC", "integer"]}
mutate {convert => ["PORT_DST", "integer"]}
mutate {strip => ["IP_SRC", "IP_DST"]}

}
output {
elasticsearch {
hosts => "10.2.51.254:9200"
index => "ipsrc_dst_portsrc_dst_data-%{+YYYY.MM.dd}"
template_name => "ipsrc_dst_portsrc_dst_data"
}
stdout {}
}
```

Al guardar este archivo en la carpeta de configuración de Logstash y este estará siempre pendiente de nuevas entradas al archivo indicado en el input para leerlas, procesarlas y exportarlas. El funcionamiento de lectura es similar comando *tail -f file*.

6.2.2. Elasticsearch

ElasticSearch es un poderoso motor de búsqueda y análisis de datos, desarrollado bajo código abierto. Esta herramienta nos permite almacenar, buscar y analizar grandes volúmenes de datos rápidamente y casi en tiempo real y en general es usada por otras aplicaciones de más alto nivel como es el caso de Kibana.

Alguna de las partes estructurales de ElasticSearch se describen a continuación:

Cluster Un cluster es una conjunto de uno o más nodos que juntos mantienen las datos y proporciona capacidades de indexación y búsqueda en todos los nodos. Un cluster tiene un identificador con un único nombre , que por defecto es “elasticsearch”. El nombre es importante porque un nodo sólo puede ser parte de un cluster y este se une por su nombre.

Nodo Un nodo es un servidor que es parte de un cluster, almacenando los datos y participando en la búsqueda e indexación del cluster. Al igual que el cluster, un nodo es identificado por un nombre único el cual por defecto es asignado con un conjunto de caracteres random que se asignan cuando se crea el nodo. Asignarle un nombre es importante solo para motivos de administración cuando se trabaja con más de un servidor por cluster. Un nodo por defecto se une al cluster “elasticsearch” a no ser que se

fije de manera particular.

Index Un index es un conjunto de documentos que tienen características comunes, de esta manera se pueden tener index para cada una de las salidas del preprocesador y otra para las notificaciones de SNORT. Un index es identificado por un nombre (todo con minúsculas) y este nombre es usado cuando se realizan operaciones de indexación, búsqueda, actualización o eliminación el un documento documento.

Type Dentro de n index, pueden existir uno o más *Type*, cada uno es estos son una partición o categoría del index y es definido por el usuario (estas son las separaciones que se realizan en la etapa de filtrado en Logstash). De alguna manera podemos decir que son las columnas de una matriz en donde se almacenan los datos. Así dentro de un index podemos encontrar tantos documentos como se desee.

Shards Un index puede almacenar mucha información, pero la cantidad de datos podrían exceder las capacidades de la máquina que lo aloja o disminuir el rendimiento en las búsquedas. Para ello ElasticSearch creo los Shards, que son subdivisiones de un index, las cuales son completamente funcionales e independientes del index. Las búsquedas en los distintos Shards son totalmente transparentes para el usuario, para este solo existe el index.

Document Un *Document* es la unidad básica que puede ser indexada. Estas unidades son expresado (JavaScript Object Notation).

Comando útiles

Mientras ElasticSearch este activo podemos manipular las estructuras con algunos comando, ya sea por una terminal o por el mismo navegador.

Para ver el estado de los cluster:

```
curl 'url:9200/_cat/health?v'
```

Para revisar los nodos que hay los cluster

```
curl 'url:9200/_cat/indices?v'
```

Para crear un index:

```
curl -XPUT 'url:9200/customer?pretty'
```

Para borrar algún nodo:

```
curl -XDELETE 'url:9200/customer?pretty'
```

Archivo de configuración

ElasticSearch tiene un archivo de configuración con formato YAML y se puede encontrar por el nombre de elasticsearch.yml.

Algunas de los parametro de interes son

```
network.host: 192.168.0.114
```

donde hay que fijar la direccion IP en al cual esta instalado ElasticSearch, el puerto de comunicacion para hacer las operaciones sobre ElasticSearch

```
http.port: 9200
```

y las rutas donde se almacenan lo datos y los log path

```
path.data: /path/to/data

path.logs: /path/to/logs
```

.

6.2.3. Kibana

Kibana es una plataforma de análisis y visualización de código abierto diseñado para trabajar con ElasticSearch. Este permite buscar, ver e interactuar con los datos almacenados en ElasticSearch, así fácilmente se pueden construir gráficos, tablas o mapas con estos datos.

Kibana hace fácil entender grandes volúmenes de datos, es de fácil uso y basado en un navegador se pueden crear rápidamente dashboards acorde a las necesidades personales.

Para hacer funcionar kibana en funcionamiento solo se requiere configurar los *index* que se desean explorar. Para ello hay que ingresar en el navegador e ingresar al dominio u IP donde se aloja Kibana http://DOMAIN.com:5601, ir a la sección Settings y agregar un patrón de index para uno o más de los index en ElasticSearch. Por defecto Kibana

trabaja con ElasticSearch y Logstash por eso usa el patrón *logstash-**, El * es usado para tomar todos los index que comienza con ese patrón. Por último Kibana solicita cuál será el campo que se usará por timestamp, por defecto ElasticSearch crea un timestamp cuando ingresa un dato y se recomienda usar esta porque está optimizado para funcionar con este. Hacer click en crear y ya esta listo par funcionar.

Kibana al igual que ElasticSearch usar un archivo de configuración con el formato YAML, llamado kibana.yml, donde el principal parametro de configuracion es el puerto y la IP donde esta elasticsearch.

Metodos de busqueda

Para buscar dentro de los datos se usan consultas con el formato de ElasticSearch, que ayudan a explorar lo que ocurre o para generar análisis particulares. Las búsquedas se basan en los campos Type de los index. A continuación se exponen algunas formas de básicas de búsqueda.

Si queremos buscar en el campo ciudad palabras que contengan los campos Valparaíso o Viña, ocupamos el operador *OR* aunque por defecto si se omite cualquier operador es este el que se usa. De esta manera podemos hacer la consulta de estas 2 formas: `ciudad: valparaiso OR viña` o `ciudad: valparaiso viña`. También se puede usar el operador AND para interceptar las búsquedas.

Por el contrario si queremos buscar una frase exacta por ej. en el campo nombre, se requiere usar la siguiente sintaxis `nombre:"john smith"`.

Si queremos buscar todos los elementos que contienen un patrón en común, se usa * para reunir todas las variaciones de ese patrón y se usa con la siguiente sintaxis `book: math*`.

Además podemos combinar cada una de estas sintaxis para realizar búsquedas más precisas. Para más detalles en las formas de búsqueda ir a https://www.elastic.co/guide/en/beats/packetbeat/current/_kibana_queries_and_filters.html.

Una ejemplo de como se ve el dashboard se puede encontrar en la figura 6.3.

6.3. Configuración de la Red

El sistema completo está implementado en la Red de Computadores del Departamento de Electrónica de la UTFSM. Originalmente se pensó una implementación sobre el fire-

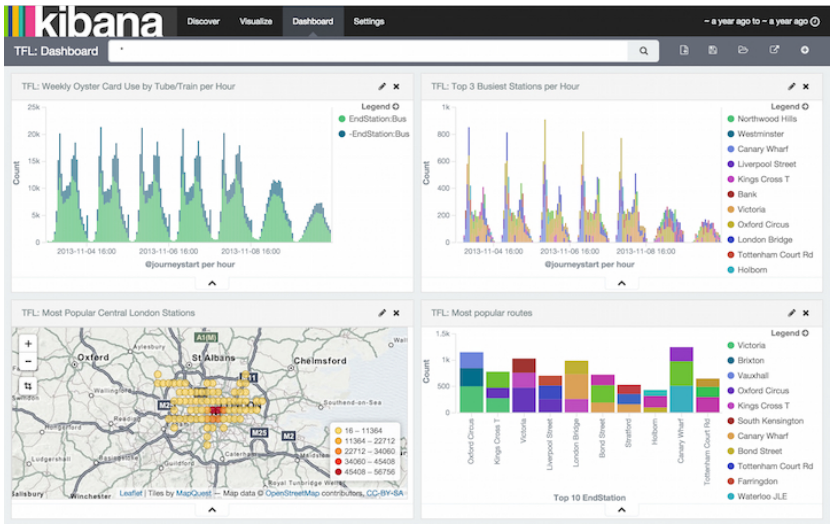


Figura 6.3: Dashboard Kibana

CPU	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz.
nCores	4.
SO	Centos 7.
Memoria	4 GB.
Almacenamiento	500 GB.
Conectividad	82579LM Gigabit Network Connection. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller.

Tabla 6.2: Especificaciones técnicas equipo SNORT AD

wall que operaba con FreeBSD pero se encontró una solución alternativa que es menos invasiva a través del switch. Para ello se hicieron configuraciones especiales en el switch que permitiera analizar lo que pasa en la red sin intervenir activamente sobre el tráfico de la red, esto se hace reflejando todo el tráfico de una boca J del switch hacia otra boca K, esta última toma todo el tráfico de entrada y salida de J. Esta configuración es conocida como *span mirror*, una representación gráfica de este esquema se muestra en la figura 6.4.

El equipo usado para analizar los paquetes corre en Centos 7 con su versión mínima de instalación donde se ejecuta SNORT 2.9.7.3 y logstash 2.3. Las especificaciones técnicas del equipo que se usó se muestran en la tabla .

En el caso de la visualización su usa un Notebook Dell funcionando con Elementary OS, una distribución basado en Ubuntu 14.04 donde se corre ElasticSearch y Kibaba. En la tabla 6.3 se muestran las características técnicas del equipo usado.

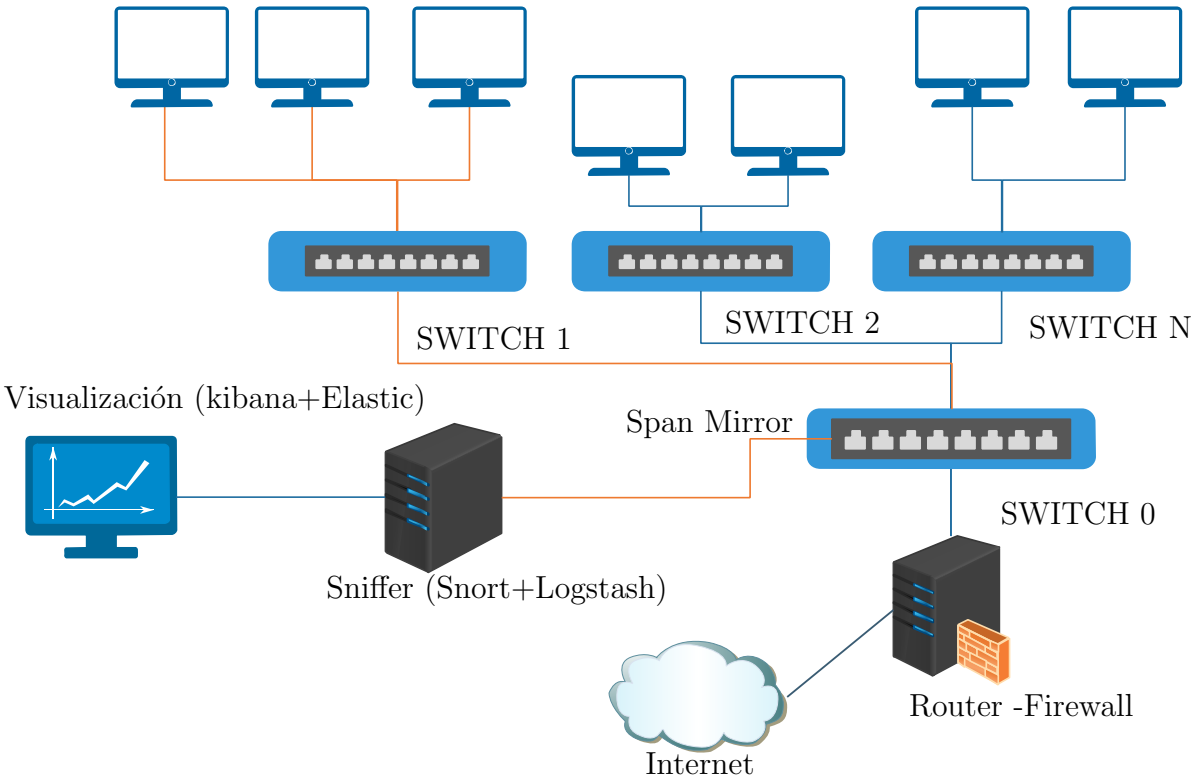


Figura 6.4: Estructura de de la red en la implementación

CPU	Intel(R) Core(TM) i5-2467M CPU @ 1.60GHz .
nCores	2.
SO	Elementary OS.
Memoria	4 GB.
Almacenamiento	160 GB SSD.
Conectividad	Wifi - Intel Corporation Centrino Advanced-N 6230

Tabla 6.3: Especificaciones técnicas equipo SNORT AD

Capítulo 7

RESULTADOS EXPERIMENTALES

En esta sección se desarrollan las pruebas al funcionamiento del algoritmo en su primera etapa, que consiste en la detección de los Deltoides más significativos. De esta manera a través de las distintas series de tiempo ver el comportamiento de ataques clásicos como DOS, Port Scan y Brute Force. La idea central acá es analizar bajo qué rangos las diferencias pueden ser consideradas como sospechosas y en lo posible poder clasificar bajo estos comportamientos que tipo de ataque es el que se está tratando de perpetuar.

Además se presenta el uso de recursos involucrados en el proceso de detección y visualización de los datos, midiendo el uso de la CPU y la Memoria RAM asociadas los equipos donde está instalado Snort, Logstash, ElasticSearch y Kibana.

7.1. Pruebas

Como se explicó en las primeras sección hay diferentes tipos de ataques, los cuales se pueden agrupar en las categorías *Scanning Attacks* o *Probing*, *Denial of Service (DoS)* y *System Penetration*. Para cada una de ellas usaremos una herramientas de test que permita realizar ataques de cada tipo y ver así el comportamiento de los Deltoides.

Todos los gráficos generados para estas secciones fueron obtenidas de Kibana, para tiempos de interés del ataque. Se pueden apreciar algunos desfase en las gráficas debido a que la visualización es remota y los datos podrías llegar en órdenes diferente y el tiempo usado para graficar es la fecha de inserción del dato en elasticsearc (limitación del sistema). Lo colores usados en las gráficas corresponden a características espaciales: el rojo indica el deltoide de mayor variación, el azul corresponde al deltoide de menor variación y el verde corresponde a la suma de los valores absolutos de los deltoides, todos estos en la ventana de tiempo de visualización, que no necesariamente corresponde a la ventana de tiempo de captura.

7.1.1. Scanning Attacks

Los *Scanning Attacks* son el primer intento de penetración en la seguridad de los sistemas, los cuales buscan la mayor cantidad de información de los servicios disponibles en los servidores. Una de las características de este tipo de ataque es que envía una gran cantidad de paquetes por unidad de tiempo y la carga de estos paquetes suele ser 0, porque lo único que buscan es ver si las respuesta a las distintos tipos de servicios está activo o no.

Una de las herramientas más usada para medir la seguridad de las redes respecto, el cual realizar un escaneo de puerto y redes es **Nmap**. Este es un software libre que trabaja con los paquetes en su versión raw, con la cual se puede obtener información de un host disponible, que tipo de servicios tiene disponibles (con nombre y versión), con que SO opera, que tipo de firewall, entre otras características. Las opciones que contiene son múltiples y sirven para realizar diferentes formas de escaneo a las redes. A continuación mostraremos algunas de sus formas y como estas se reflejan en este algoritmo.

nmap -sP 200.1.17.0/24 La opción -sP de nmap permite buscar servidores que estén activos y funcionando, saltando la el escaneo de puertos. La opción /24 revisa toda la sub-red, desde la IP 200.1.17.0 a las 200.1.17.255. De esta manera paquetes irán desde una misma fuente a múltiples ip destino.

nmap -sV 200.1.17.201 -A La opción -sV revisa qué aplicación hay detrás de cada servicio junto con la versión de esta, y la opción -A incluye todo el detalle de la consulta.

nmap -sS 200.1.17.201 La opción -sS realiza un escaneo de puerto pero de manera sigilosa con un SYN scan.

Los resultados de estos tres test se muestran en las figuras 7.1, 7.3 y 7.5 donde se puede observar el comportamiento de cada una de ellas. Para el primer caso correspondiente al primer tercio del los gráficos no se registra nada, esto debido a que la cantidad de paquetes fue demasiado poco como para registrar cambios significativos. Sin embargo las otras 2 pruebas si muestran resultados considerables, en ambos caso su aporte en la serie de tiempo de IP SRC e IP DST es al menos en una de las muestras cercana a 1000 paquetes. En el caso de las series de tiempo de IP SRC/DST PORT SRC solo la tercera prueba marca un registro porque todos los paquetes salieron de un mismo puerto fuente. En la serie de tiempo IP SRC/DST PORT DST como es de esperarse no se registran mayores cambios (menores a 100 paquetes), porque se está realizando un barrido de puertos destino. Por último en las gráficas de IP SRC/DST se pueden observar claramente ambos test, con variaciones cercanas a 1000 paquetes y la serie de tiempo IP SRC/DST PORT SRC/DST no registra ningún cambio significativo (todas menores a 20 paquetes), que muestra que no existieron conexiones permanentes en los mismos puertos e IPs.

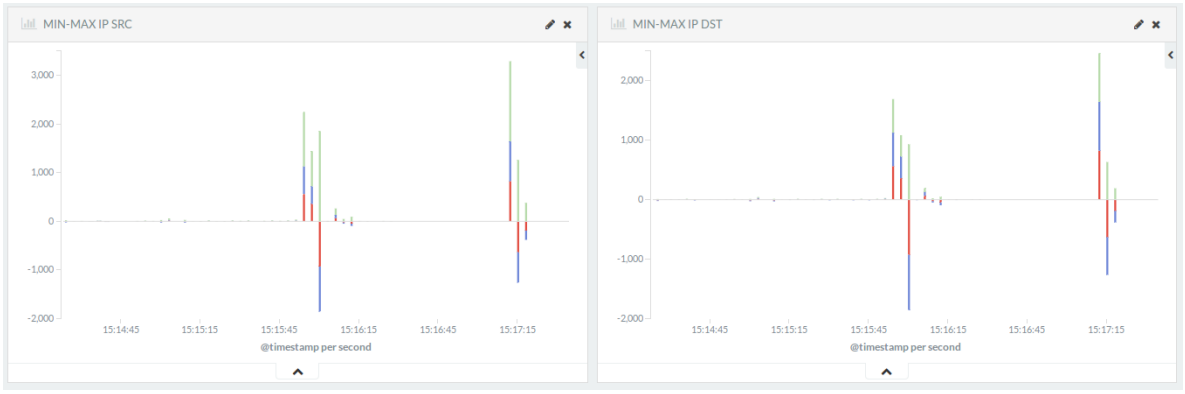


Figura 7.1: Deltoides generados por Nmap en IP SRC e IP DST

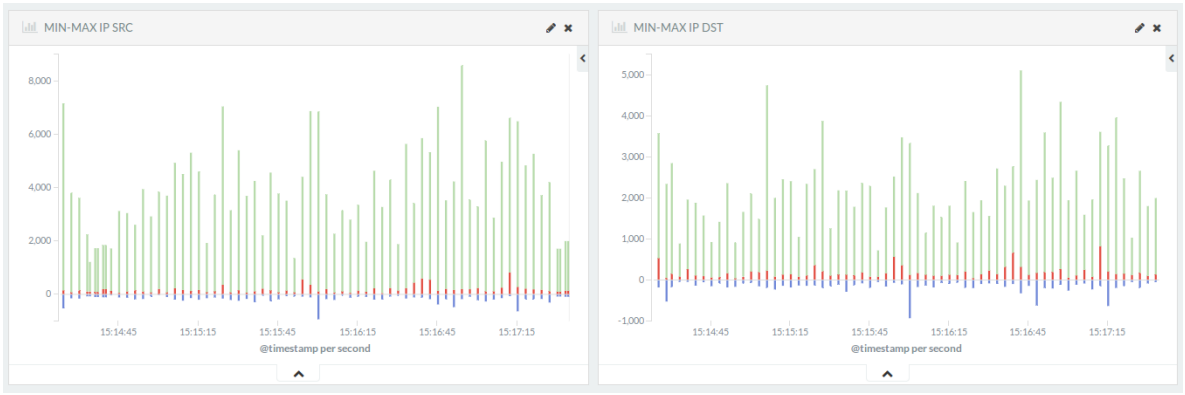


Figura 7.2: Deltoides totales en IP SRC y IP DST

Las figuras 7.2, 7.4 y 7.6 muestran la salida de todos los deltoides donde se puede apreciar como los últimos 2 test de destacan sobre el resto de los deltoides.

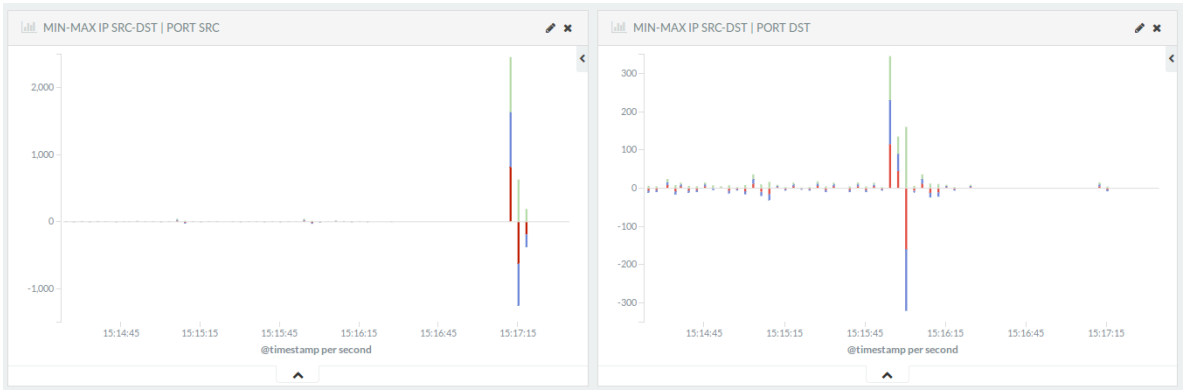


Figura 7.3: Deltoides generados por Nmap en IP SRC/DST PORT SRC o DST

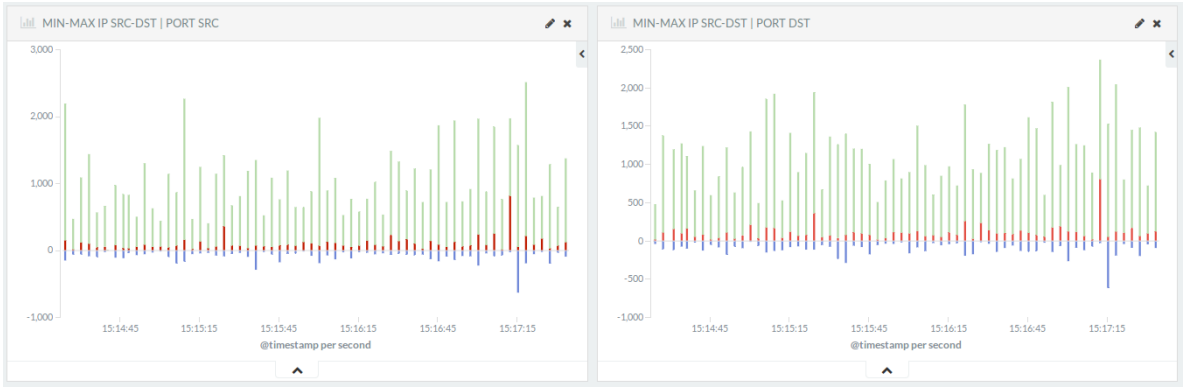


Figura 7.4: Deltoides totales en IP SRC/DST PORT SRC o DST

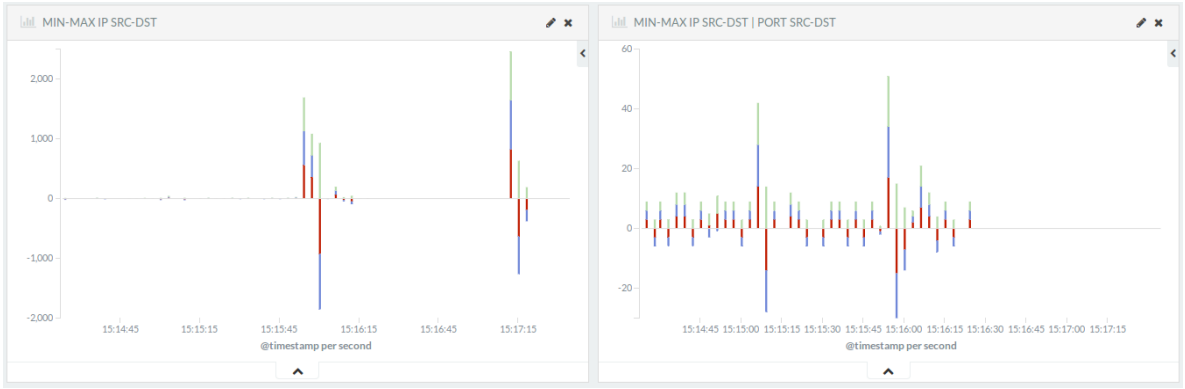


Figura 7.5: Deltoides generados por Nmap en IP SRC/DST e IP SRC/DST PORT SRC/DST

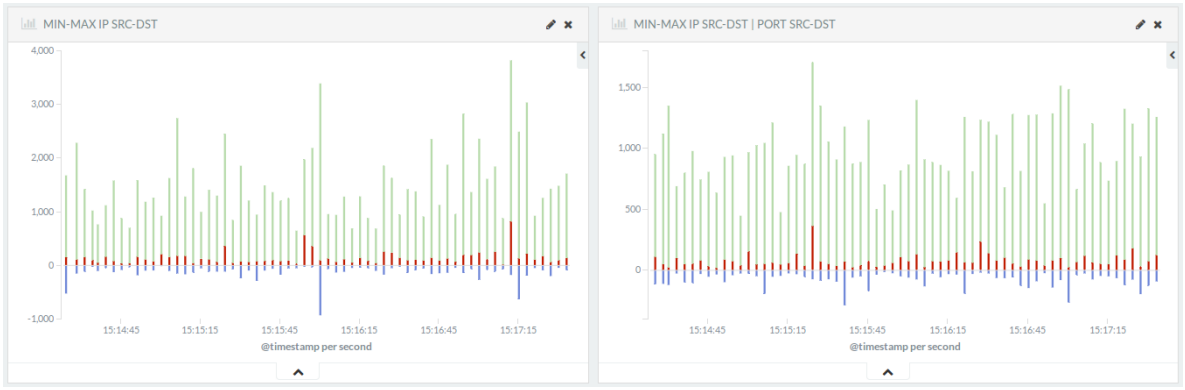


Figura 7.6: Deltoides totales en IP SRC/DST e IP SRC/DST PORT SRC/DST

7.1.2. DoS

Los ataques de denegación de servicio colapsan el uso de recurso de un servicio realizando miles de consultas en muy poco tiempo lo que genera una degradación en los tiempos de respuesta del servidor y en casos más graves lo deja fuera de servicio.

Hay muchas formas de realizar este tipo de ataque, en particular acá se utiliza hping3 que es una herramienta que nos permite crear y leer paquetes y así realizar diferentes pruebas de seguridad desde, escaneo de puerto como también DDoS/DoS. Algunas de las múltiples formas de utilización son descrita a continuación:

hping3 -p 80 –flood 200.1.17.10 Con la opción -p se especifica el puerto al cual se realizarán las consultas, con –flood se envían cientos de paquetes en un segundo y por último se especifica la ip a la cual se realizan las consultas. Con esto logramos realizar un intento de DoS, desde una IP fuente a una Ip destino.

El resultado de este test se observa en las figuras 7.7, 7.9 y 7.11, donde se puede observar el comportamiento de este en las distintas series de tiempo. Para el primer caso donde se observan solo la IP SRC o IP DST queda en evidencia que se están enviando una enorme cantidad de paquetes con diferencias superiores a los 20.000 paquetes, que es coherente con lo que se espera de una denegación de servicios. Por otro lado se observa en el segundo gráfico que la serie de tiempo IP SRC/DST PORT SRC no registra grandes variaciones porque las puerto fuentes son aleatorios en este test, sin embargo en el caso de IP SRC/DST PORT DST se evidencia que todo este flujo de paquetes va a parar a un puerto destino específico que en este caso es el 80, con variaciones mayores a 20.000 paquetes. El caso de IP SRC/DST es solo un caso particular del caso anterior con los mismos resultados. Por último la serie de tiempo IP SRC/DST PORT SRC/DST no registra grandes cambios como era de esperarse porque el mayor tráfico se esta realizando a un solo puerto.

En las figuras 7.8, 7.10 y 7.12 se muestra el conjunto completo de deltoides capturados cuando se realizó el test donde se observa claramente el alto porcentaje de paquetes provocado por el test. Además se observa que debido a la gran cantidad de paquetes producto de la denegación de servicio, se generarán ventanas de tiempo donde no hay presencia de deltoides, esto se debe a que el umbral de detección crece proporcional a la cantidad de paquetes en la ventana de tiempo por lo que difícilmente un tráfico normal podría generar una diferencia de tráfico que supere el umbral.

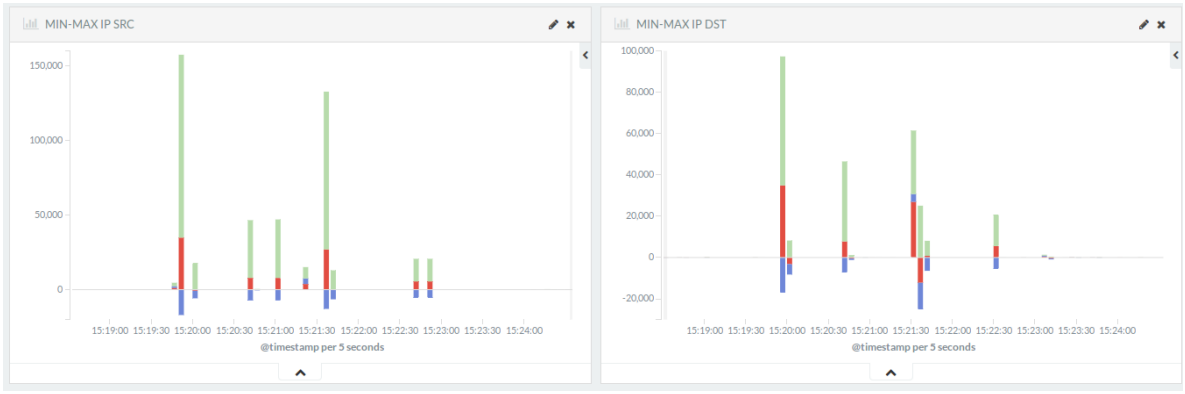


Figura 7.7: Deltoides generados por Hping3 en IP SRC e IP DST

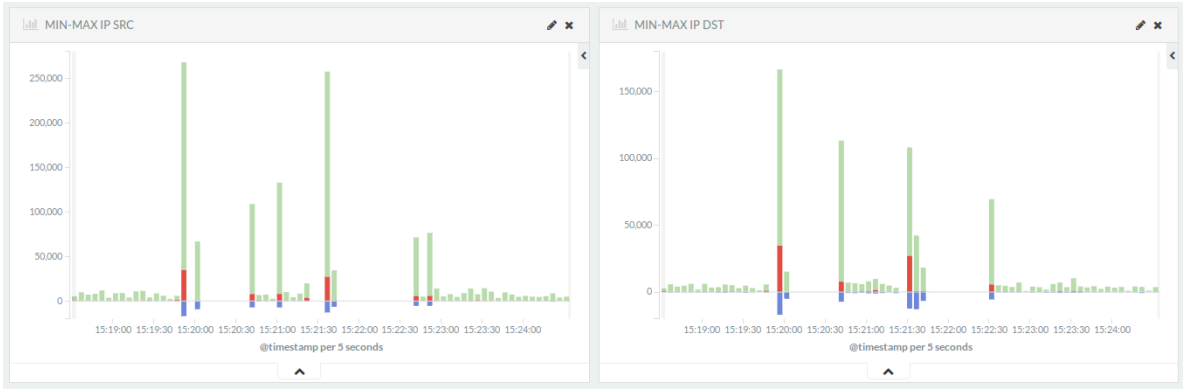


Figura 7.8: Deltoides totales en IP SRC e IP DST

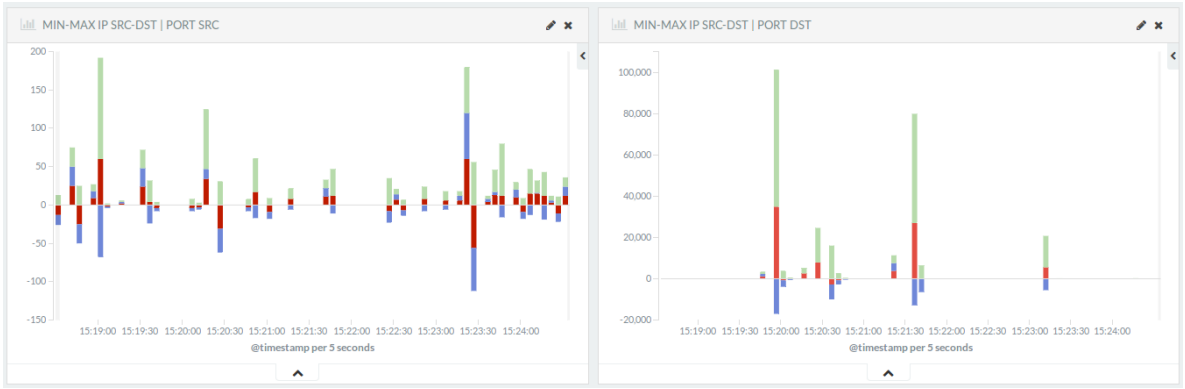


Figura 7.9: Deltoides generados por Hping3 en IP SRC/DST PORT SRC o DST

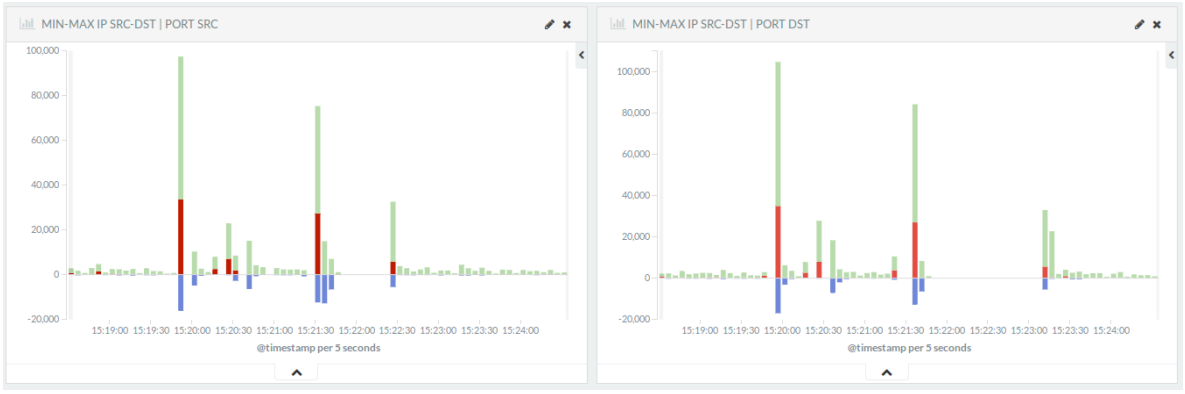


Figura 7.10: Deltoides totales en IP SRC/DST PORT SRC o DST

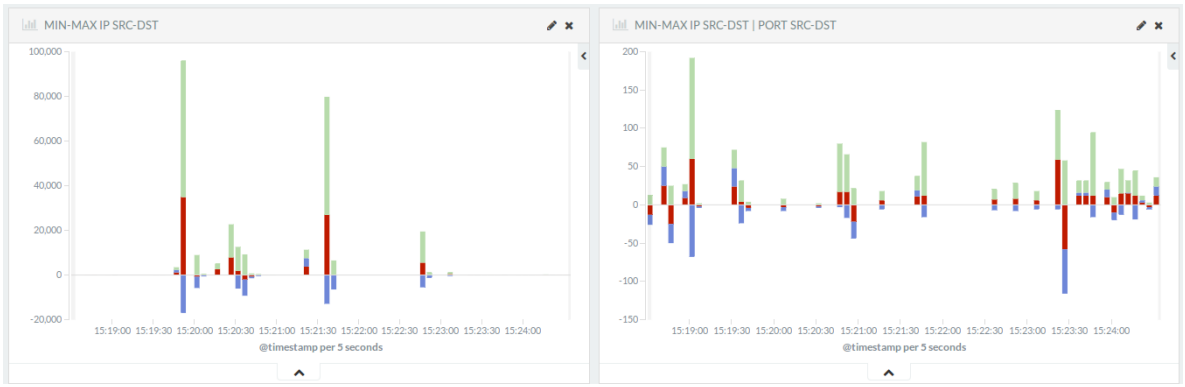


Figura 7.11: Deltoides generados por Hping3 en IP SRC/DST e IP SRC/DST PORT SRC/DST

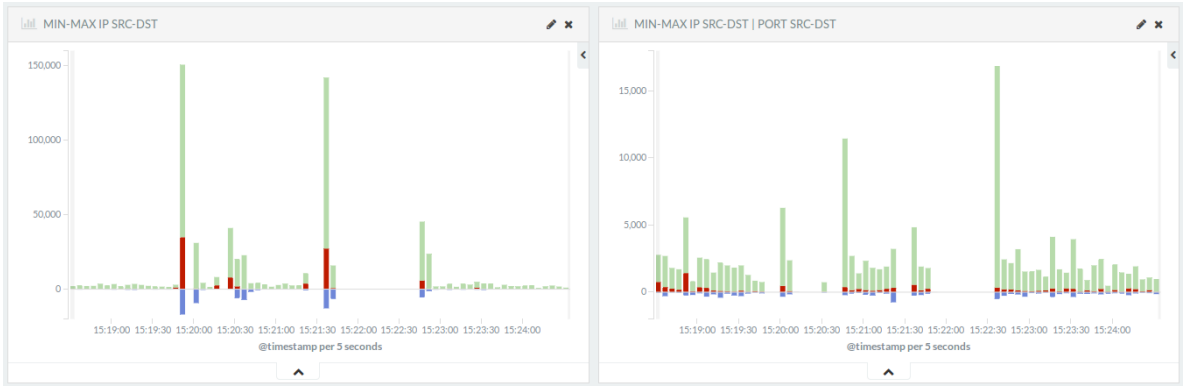


Figura 7.12: Deltoides totales en IP SRC/DST e IP SRC/DST PORT SRC/DST

7.1.3. System Penetration

En este caso la idea del ataque es tratar de tomar control de la máquina remota, rompiendo con la seguridad de alguna manera. Una de las formas es intentar acceder a través de SSH el control del usuario root con una base de datos de password e iterar hasta intentar dar con la clave, más conocido como un ataque de fuerza bruta.

La herramienta usada en este caso es *ncrack* e *hydra*, diseñadas especialmente para crackear passwords y probar la seguridad de las cuentas en una red, estas toman un archivo de texto con el diccionario de password e iteran sobre un un usuario del servidor para intentar dar con el control del sistema.

ncrack -p 22 –user root -P john.txt 200.1.17.195 La opción -p indica el puerto al que se desea hacer el ataque, –user indica el nombre del usuario al cual se intenta acceder, -P indica que se usará un fichero con passwords que en este caso es john.txt y finalmente se asigna la IP a la cual se desea realizar el ataque.

hydra -l root -P john.txt 200.1.17.201 ssh Para el caso de hydra se usa -l para indicar el usuario al que se tomar el control, -P indica que se usará un fichero con passwords que en este caso es john.txt, a continuación se asigna la IP a la cual se desea realizar el ataque y finalmente va el tipo de servicio donde se realizará el ataque.

Los resultados de estos test son mostrados solo en los obtenidos en el primer test, debido a que el comportamiento observado en ambos fue el mismo por lo que se puede generalizar las conclusiones. Las figuras 7.13, 7.15 y 7.17 muestran el comportamiento en las distintas series de tiempo.

Para las 6 series de tiempo se puede observar un comportamiento similar, lo más significativo es que los deltoides están presentes en todas series de tiempo, el problema radica que las diferencias son muy pequeñas lo que la hace difícil de detectar sólo con esta primera etapa del algoritmo. Como se observa en las figuras 7.14, 7.16 y 7.18 el aporte de estos deltoides no es significativo con respecto al total de deltoides presentes al momento de realizar el test. Una opción para revertir el problema del pequeño tamaño de los deltoides sería probar seria usar la cantidad de bytes en vez de la cantidad de paquetes lo cual podría arrojar mejores resultados porque a diferencia de los otros ataques acá siempre se está transfiriendo data en el payload.

7.1.4. Descargas

Existen algunas acciones que podrían tener comportamiento similares a los de los ataque ante vistos, los que podrían generar falsos positivos. Para ver alguno de estos casos analizamos cómo se comporta el sistema frente a un test de velocidad y a una descarga

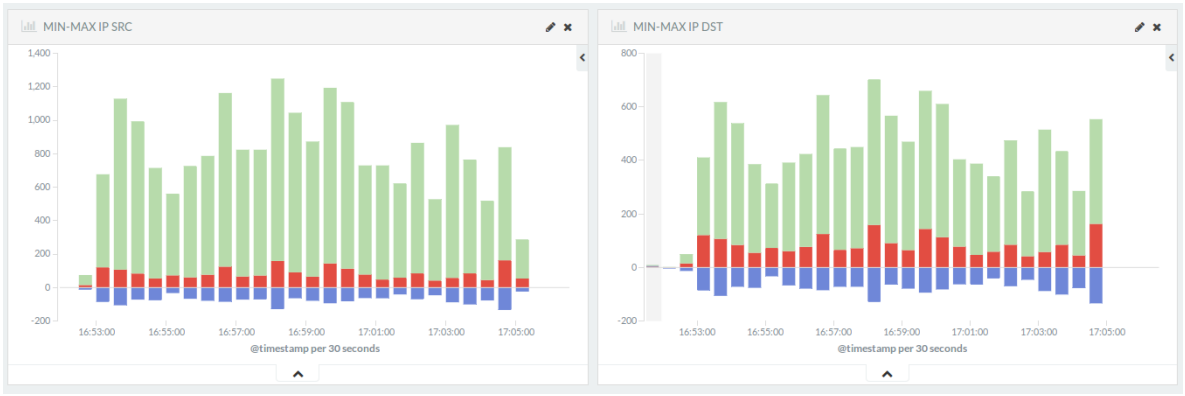


Figura 7.13: Deltoides generados por ncrack en IP SRC e IP DST

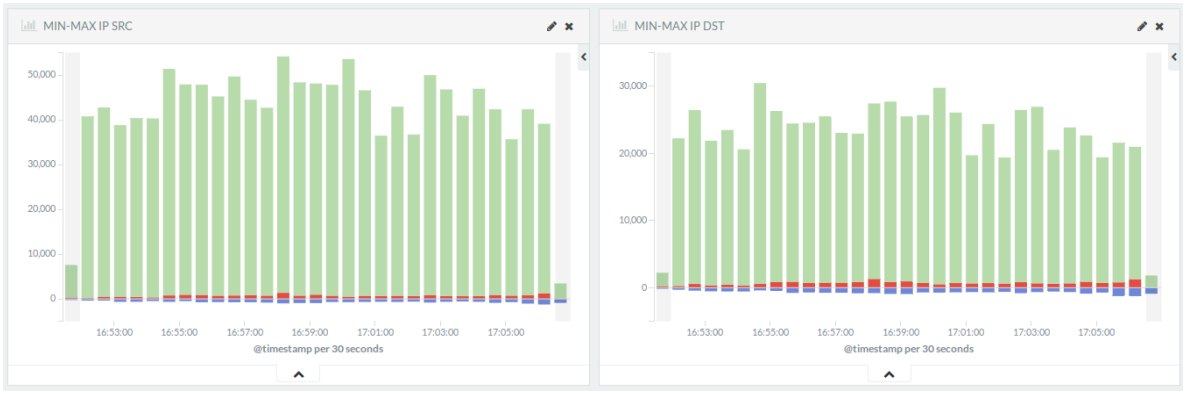


Figura 7.14: Deltoides totales en IP SRC e IP DST

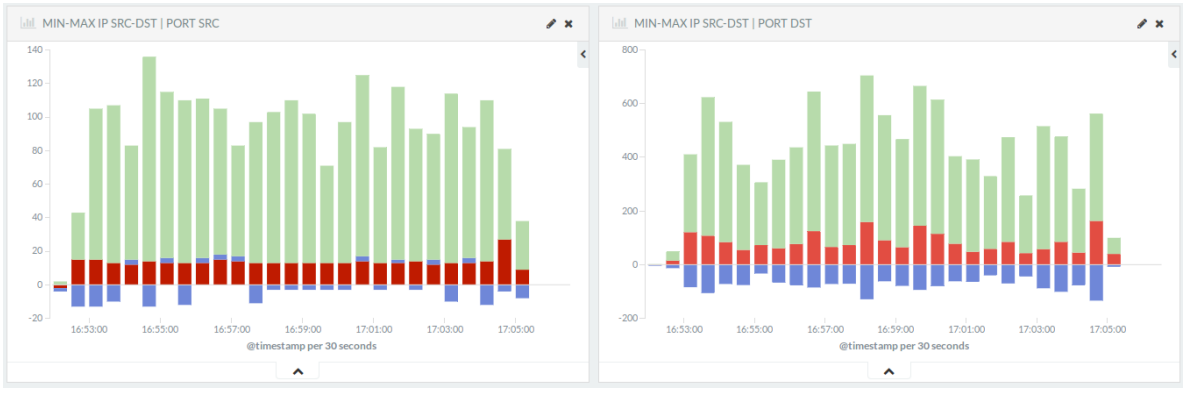


Figura 7.15: Deltoides generados por ncrack en IP SRC/DST PORT SRC o DST

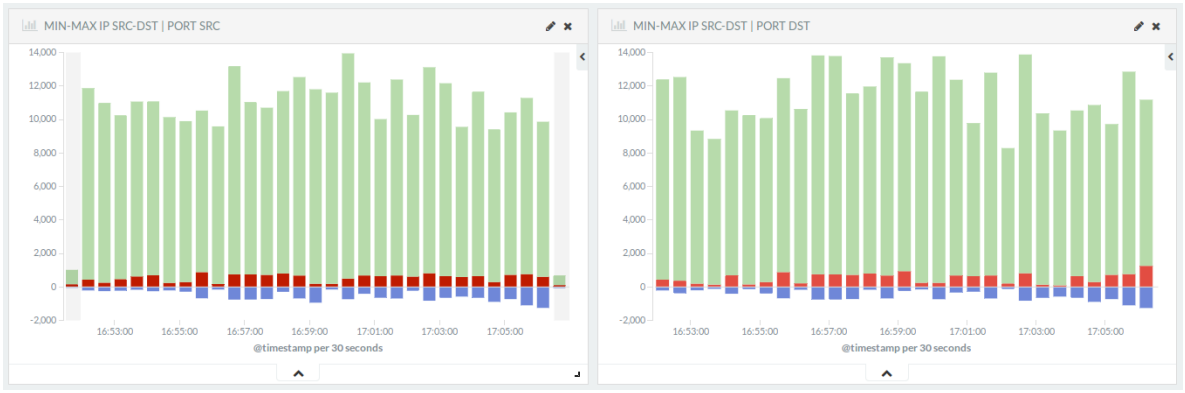


Figura 7.16: Deltoides totales en IP SRC/DST PORT SRC o DST

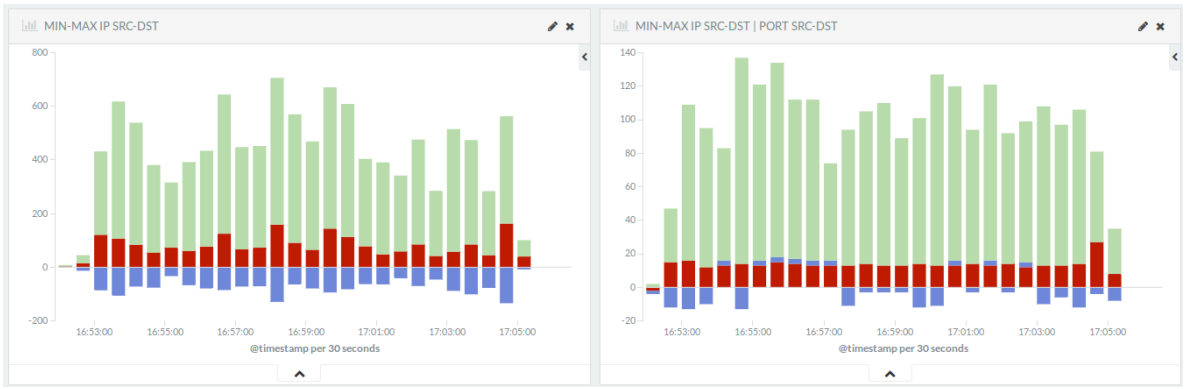


Figura 7.17: Deltoides generados por ncrack en IP SRC/DST e IP SRC/DST PORT SRC/DST

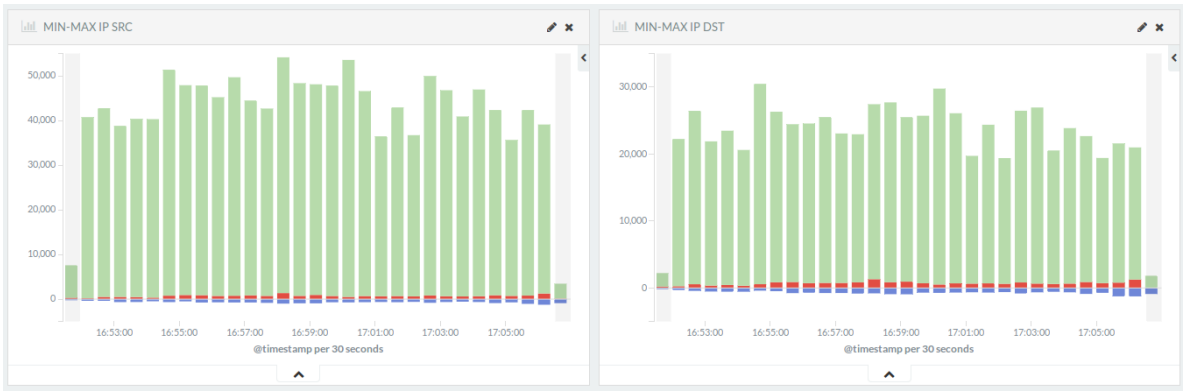


Figura 7.18: Deltoides totales en IP SRC/DST e IP SRC/DST PORT SRC/DST

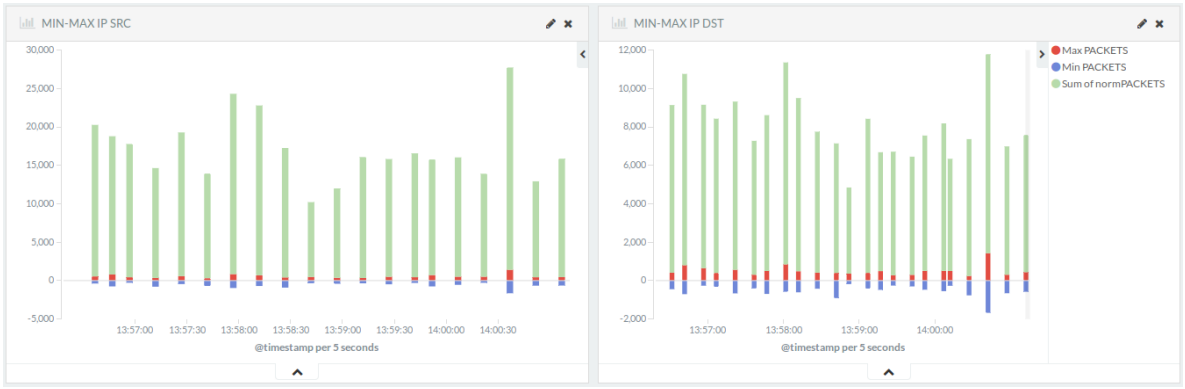


Figura 7.19: Deltoides totales en IP SRC e IP DST para trafico con descargas

de un paquete dentro de la misma red.

Test de velocidad Ookla Para hacer el test de velocidad se usa el servicio que provee http://www.speedtest.net/ para generar un gran trafico y ver como este afecta en el algoritmo.

Descarga directa Con el afán de ver el comportamiento de una descarga directa de un archivo dentro de la red de la universidad y ver como este afecta al sistema, se usa el repositorio de Latex que tiene el Departamento de Informática http://ftp.inf.utfsm.cl/pub/tex-archive/systems/win32/miktex/setup/basic-miktex-2.9.6022-x64.exe con el comando wget.

En ambos casos no se registraron variaciones significativas en ninguna de las series de tiempo en cuanto a la cantidad de paquetes, siendo una buen comportamiento para no generar falsos positivos por el algoritmo. Sin embargo en los gráficos se puede visualizar un deltoide casi al final del tiempo, pero se reviso y era el robot BaiDuSpider que explora sitios web para el buscador Chino BaiDu por lo que se descarta de las mediciones que se realizaron en ese ventana de tiempo.

Con los resultados obtenidos se puede ajustar el umbral de salida, considerando que las dos primeros tipos de ataques presentaron cambio realmente significativos, con variaciones superiores a los 800 paquetes en 3 seg. que la ventana de tiempo que se usó para el cálculo de los deltoides.

7.2. Rendimiento del Sistema

Otro de los factores importantes a la hora de ver el comportamiento del sistema de detección de intrusiones, es el uso de los recursos físicos disponibles tanto para el sistema que procesa los datos como el que los visualiza, porque de ello depende el correcto

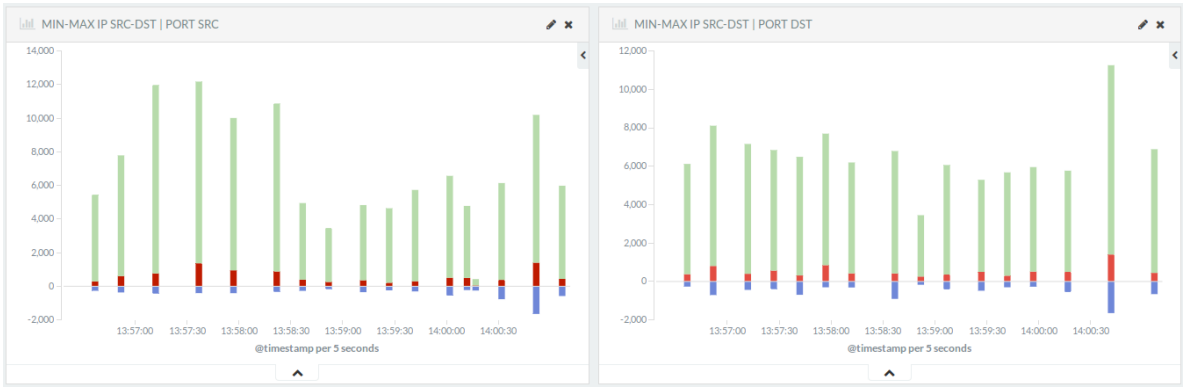


Figura 7.20: Deltoides totales en IP SRC/DST PORT SRC o DST para trafico con descargas

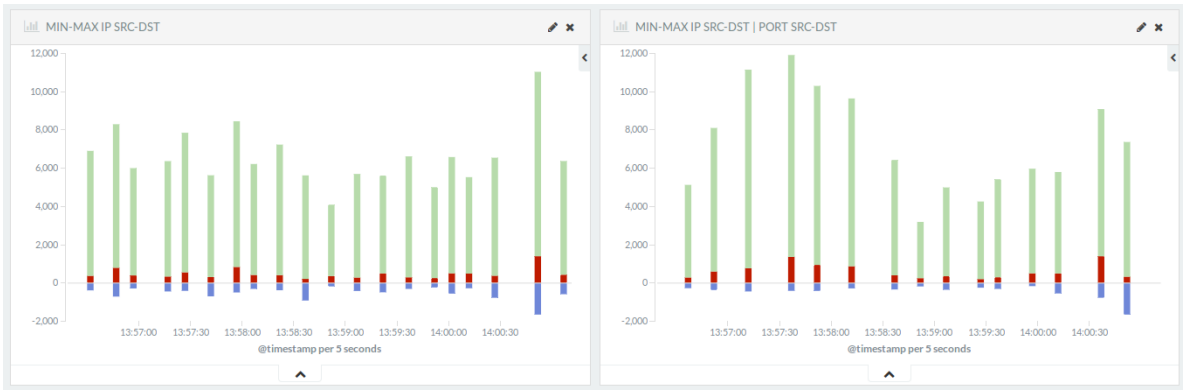


Figura 7.21: Deltoides totales en en IP SRC/DST e IP SRC/DST PORT SRC/DST para trafico con descargas

funcionamiento y detección de los ataques. Si el sistema llegara a quedar sin recursos, posibles ataque no podrían ser detectados quedando ciego frente a lo que ocurre en la red.

Todos los casos de análisis fueron realizados en paralelo el día viernes 8 de Julio, donde se registra una alta tasa de transferencia en la red analizada, generando así un análisis en una situación de uso regular.

La herramienta que se utilizo para obtener las datos de rendimiento se realizaron con *pidstat* usando las opciones '-h -r -u -p' en intervalos de 5 minutos.

A continuación se muestran las gráficas del uso de CPU y Memoria RAM para el equipo donde corre Snort y Logstash descrito en la tabla 6.2. Los valores están descritos en porcentajes respecto al total, en particular el caso de la CPU se refiere al uso de una

sola CPU.

7.2.1. Rendimiento de la detección

Snort en esta versión usa solo una CPU por lo que se registra sólo está. Así los resultados para esa CPU se muestran en la figura 7.22, donde se ve una fluctuación del uso de la CPU entre un $9 - 20\,\%$, sin grandes cambios por lo que puede garantizar una estabilidad del sistema en el uso de CPU. Para el caso del uso de la Memoria RAM no se aprecian variaciones por lo que el uso de RAM no se correlaciones con el comportamiento de la red, garantizando así su correcto funcionamiento independiente de las condiciones externas.

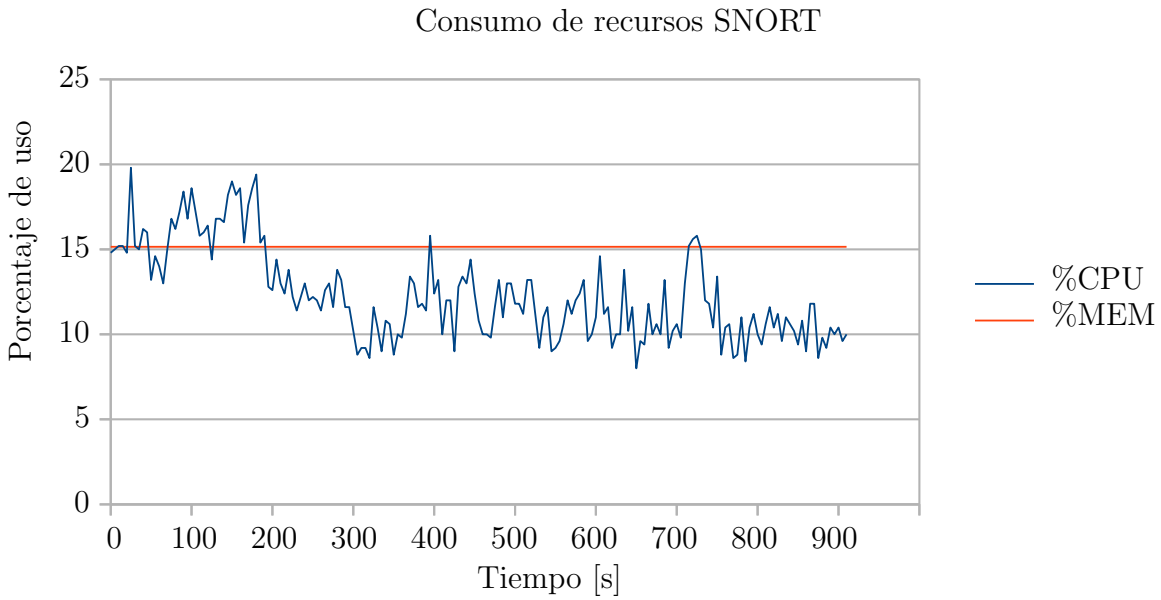


Figura 7.22: Rendimiento Snort

Para el caso de Logstash, cada fichero que se usó para almacenar las distintas series de tiempo usan un proceso diferente. Para el caso de estudio se usaron 6 series de tiempo y estas son visualizadas en el gráfico 7.23, donde podemos observar que el uso de la CPU por cada proceso es significativamente baja (cercano el $3\,\%$) con algunas excepciones donde ocurrieron peak que en el peor de los casos no superó el $18\,\%$ del uso de una CPU, debido probablemente a una cantidad alta de deltoides encontrados en una ventana de tiempo. Por otro lado el uso de la memoria RAM al igual que Snort tiene casi una nula variación siendo un poco más del $6\,\%$ del uso total de la memoria del sistema, lo que podría ser considerable si son muchos los ficheros que se deseen leer.

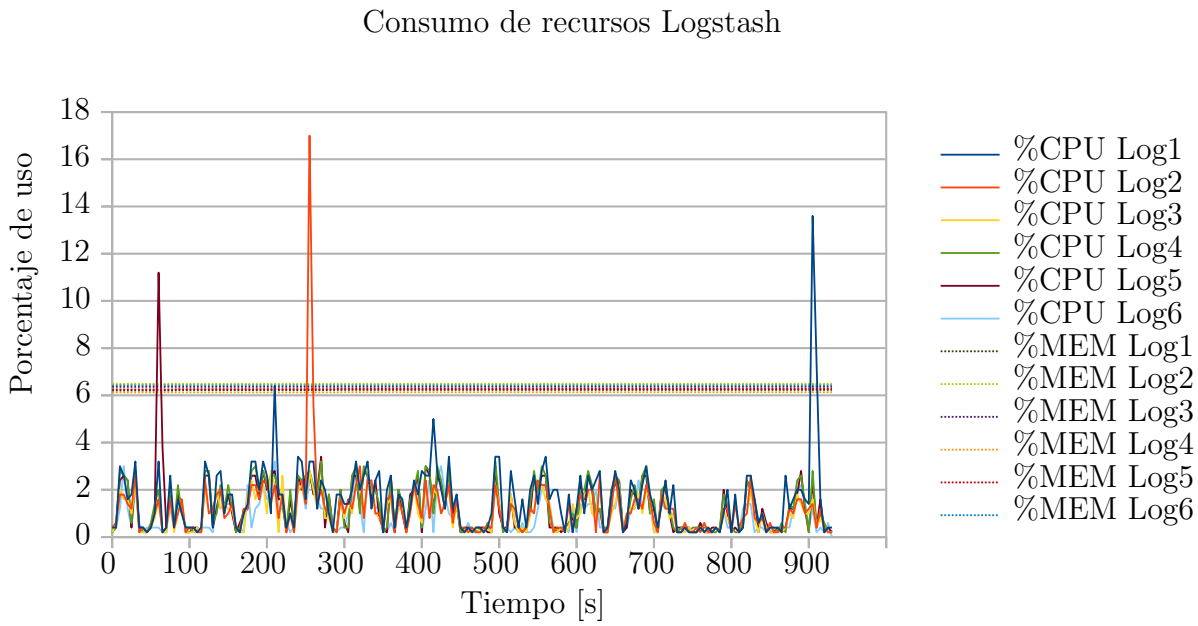


Figura 7.23: Rendimiento Logstash

7.2.2. Rendimiento en la visualización

Para la visualización se usa el notebook descrito en la tabla 6.3 y al igual que en el caso anterior los resultados son mostrados en porcentajes relativos al total.

Para ElasticSearch el consumo de recursos es mostrado en la figura 7.24 donde se observa que la CPU es significativamente bajo en los casos donde solo se reciben datos (bajo el 3 %) pero cuando se realizan búsquedas puede tener peak de consumo para procesar la consulta que se está realizando. En cuanto al uso de la Memoria RAM es constante con algunas ligeras fluctuaciones que no superan el ±0,5 % sobre el basal de 6 %.

Por último, el uso de recursos usado por Kibana se muestra en la figura 7.25. Aquí se puede observar que el uso de CPU es despreciables debido a que todo el procesamiento de los datos lo hace ElasticSearch. Sin embargo el uso de la Memoria RAM no es despreciable, dado que casi alcanza el 20 % del total del sistema. La tendencia ascendente de esta podría estar dado por algún tipo de dato cache, pero para descartar se deberían hacer mas pruebas o tiempos más prolongados de medición.

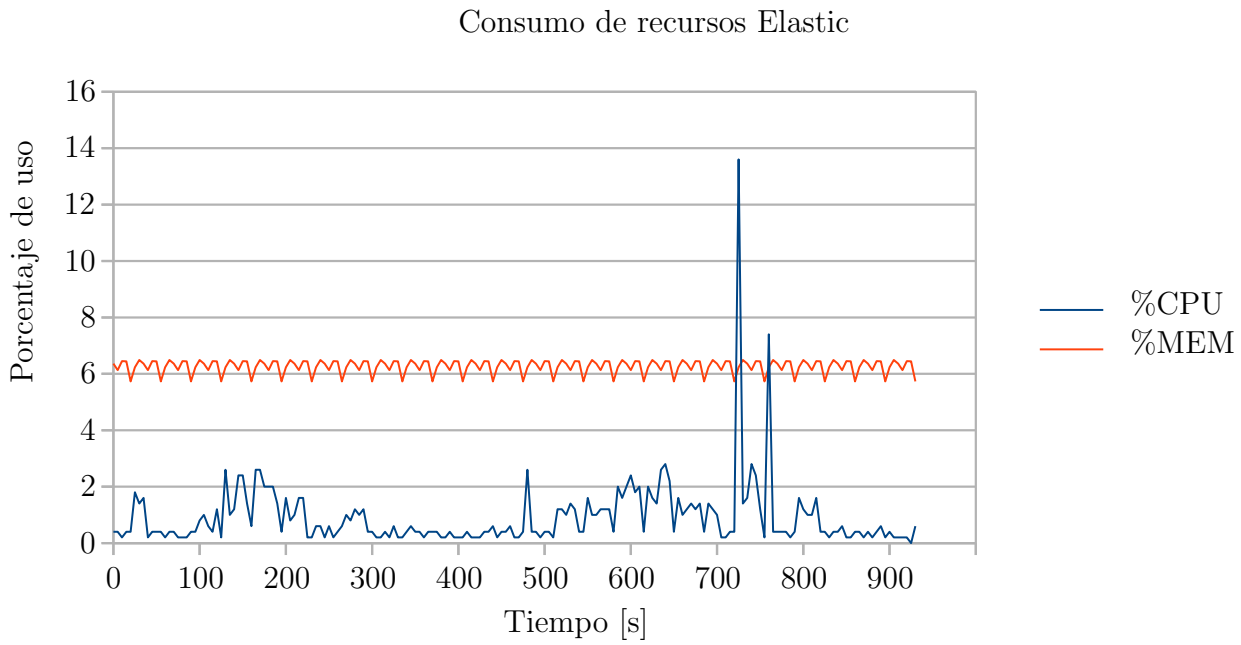


Figura 7.24: Rendimiento de ElasticSearch

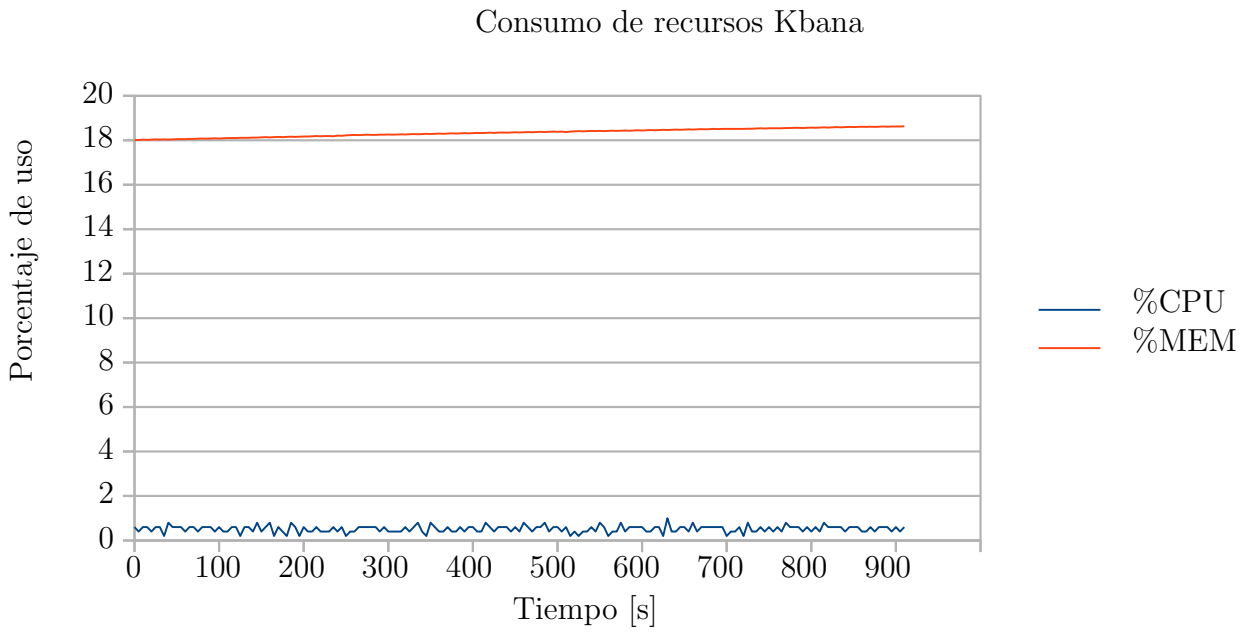


Figura 7.25: Rendimiento de Kibana

Capítulo 8

CONCLUSIONES

El amplio recorrido de este trabajo en la búsqueda de implementar un Sistema de detecciones de Intrusiones Híbrido permite visualizar la enorme cantidad de posibilidades que existen hoy para desarrollar mecanismos que ayuden a resguardar la seguridad de los sistemas informáticos, que como se vio es un tema que cada vez más toma importancia para los empresas e instituciones con grandes cantidades de tráfico e información que debe ser resguardada.

De esta manera la implementación de mecanismos o sistemas que nos brinden más información de lo que ocurre en las redes, más allá de lo que pasa en la primera barrera de seguridad (Firewall) es de gran utilidad para crear nuevas técnicas para prevenir y detectar intentos de vulneración de la seguridad del sistema. De esta manera toma gran importancia los IDS y más aún un IDS con detección en tiempo real de lo que ocurre en la red.

Si bien este trabajo no logró el objetivo final de implementar las tres partes del algoritmo propuesto, se logró dejar un cimiento que ya es útil para detectar amenazas en tiempo real que complementan el aporte de la detección de Snort a través de firmas, además de dejar abierto el trabajo para que otras personas lo tomen y puedan seguir desarrollando esta idea.

Un aporte importante de este trabajo de titulo es la documentación de como desarrollar un modulo preprocesador para Snort, lo que actualmente no existe en la red, permitiendo a facilitar y reducir los tiempos de desarrollo de personas interesadas en seguir con este trabajo.

Por otro lado cada una de las series de tiempo aporta en la detección de ataques, pero sin duda las que siempre entregan información, por lo tanto las esenciales, son las de menor resolución, IP SRC e IP DST, donde se observó que responden de manera tangible a las pruebas de seguridad que se realizaron, las otras series de tiempo aportan información que nos permite identificar con más precisión qué tipo de ataque es al que

se refieren o por el contrario no entregan información. Esto es importante porque una de las cosas que se requiere para las etapas posteriores son indicadores globales que se activen cuando ocurren fenómenos anómalos para así entregar los datos al clustering. Con los resultados obtenidos de manera experimental acá se puede garantizar que si si se pueden usar como un primer indicio de detección.

Otro factor importante que se puede rescatar de esta trabajo es que además de la detección de anomalías, también puede ser usado de manera forense, para revisar off-line situaciones extrañas que el administrador de la red detecte con las herramientas que ya usa y que ahora podría profundizar en qué fue lo que ocurrió a través de las series de tiempo en sus distintas resoluciones a lo largo del tiempo, porque estas usan muy poco espacio al tener la información resumida.

Respecto al rendimiento del sistema, el equipo de detección consume pocos recursos para estar analizando todo el tráfico que circula por el Departamento de Electrónica, pero la visualización de no es tan eficiente por lo que sería útil migrar está a un equipo con mejores características que permita reducir los tiempos de respuesta en la búsqueda de lo que pasó en las series de tiempo.

Por último se destaca que el uso del software libre como una herramienta para el desarrollo de nuevas soluciones es una fortaleza que se ve reflejada en trabajos como este, donde tener disponible el código para que cualquiera lo pueda modificar, permite a terceros solucionar problemas de interés particular sin tener que construir todo desde cero y a la vez esta nueva solución puede ser mejorada por otros que tomen interés por este trabajo.

REFERENCIAS

[1] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Comput. Commun.*, vol. 35, no. 7, pp. 772–783, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2012.01.016

[2] R. Bace and P. Mell, *NIST Special Publication on Intrusion Detection Systems*.

[3] S. Mukkamala, A. Sung, and A. Abraham, "Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools," *Department of Computer Science*, vol. 50, pp. 112–115, 2010.

[4] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, ser. DSNW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 19–24. [Online]. Available: http://dx.doi.org/10.1109/DSNW.2011.5958858

[5] Jyothsna and R. Prasad, "Article: A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, no. 7, pp. 26–35, August 2011, full text available.

[6] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," *Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden*, 2000.

[7] SourceFire, "Snort, intrusion detection system," http://www.snort.org/documents/2, accedido 1 Abril, 2015.

[8] U. Z. Ortega, *Estado del Arte Sistemas de Detección de Intrusos*, Octubre 2004.

[9] V. R. Vemuri, *Enhancing Computer Security with Smart Technology*, 2006.

[10] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 5, pp. 649–659, Sept 2008.

[11] X. Tong, Z. Wang, and H. Yu, "A research using hybrid rbf/elman neural networks for intrusion detection system secure model," *Computer Physics Communications*, vol. 180, no. 10, pp. 1795 – 1801, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465509001519

[12] X. Yu, "A new model of intelligent hybrid network intrusion detection system," in *Bioinformatics and Biomedical Technology (ICBBT), 2010 International Conference on*, April 2010, pp. 386–389.

[13] Selim, Hashem, and Nazmy, "Hybrid multi-level intrusion detection system," *International J. Computer Science and Information Security*, vol. 9, no. 5, pp. 23–29, 2011.

[14] McAfee, "Ficha técnica mcafee network security platform," http://www.mcafee.com/mx/products/network-security-platform.aspx, accedido 1 Abril, 2015.

[15] P. A. Networks, "Networks next-generation firewalls," https://paloaltonetworks.com/products/features/ips.html, accedido 1 Abril, 2015.

[16] Cisco, "Next-generation intrusion prevention system," http://www.cisco.com/c/en/us/products/security/ngips/index.html, accedido 1 Abril, 2015.

[17] T. S. Project, *Snort Users Manual 2.9.7*, 2015.

[18] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *IEEE/ACM Trans. Netw.*, vol. 13, no. 6, pp. 1219–1232, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1109/TNET.2005.860096

[19] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Network anomaly detection and classification via opportunistic sampling," *IEEE Network*, vol. 23, no. 1, pp. 6–12, January 2009.

[20] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: Global characteristics and prevalence," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03. New York, NY, USA: ACM, 2003, pp. 138–147. [Online]. Available: http://doi.acm.org/10.1145/781027.781045

[21] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 90–105, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1007730.1007731

[22] M. Ester, H. peter Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996, pp. 226–231.