ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"ADAPTACIÓN DEL IDS/IPS SURICATA PARA QUE SE PUEDA CONVERTIR EN UNA SOLUCION EMPRESARIAL"

TESINA DE SEMINARIO

Previa a la obtención de los Títulos de:

INGENIERO EN CIENCIAS COMPUTACIONALES
ESPECIALIZACION SISTEMAS DE INFORMACIN

INGENIERO EN TELEMÁTICA

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentada por:

JUAN ANTONIO ASTUDILLO HERRERA

ALBERTO ALEJANDRO JIMENEZ MACIAS

FERNANDO MANUEL ORTIZ FLORES

Guayaquil - Ecuador

2011

AGRADECIMIENTO

Agradecemos a Dios y a nuestras familias por darnos fuerza y apoyo en todo, a los profesores que nos han enseñado sus conocimientos y experiencia y a nuestros amigos que nos han apoyado en nuestra carrera universitaria.

DEDICATORIA

A Dios

A nuestros padres

A nuestros familiares

A nuestros amigos

TRIBUNAL DE SUSTENTACIÓN

_____

Ing. José Alfonso Aranda
PROFESOR DE LA MATERIA

_____

Ing. Guido Caicedo
PROFESOR DELEGADO DEL DECANO.

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesina de Grado, nos corresponden exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL"

(Reglamento de Graduación de la ESPOL)

_____

Juan Antonio Astudillo Herrera

_____

Alberto Alejandro Jiménez Macías

_____

Fernando Manuel Ortiz Flores

RESUMEN

Los Sistemas de Prevención y Detención de Intrusos (IPS/IDS) proporcionan un nivel adicional de seguridad en las redes de datos cubriendo agujeros de seguridad que los firewall simplemente no pueden. La mayoría de estos sistemas son pagados y costosos, por lo que resultan muy difíciles de implementar en pequeñas y medianas empresas.

Suricata es un motor IPS/IDS de código abierto bajo licencia GPLv2 relativamente nuevo pero con muy buenas características siendo la más importante su arquitectura multi-hilos. El objetivo principal del proyecto es montar una solución IPS/IDS empresarial con Suricata como motor de detección.

En primer lugar se definió el estado de arte del proyecto Suricata y se estudió los requerimientos necesarios para diseñar una solución IPS/IDS de uso empresarial que no se quede tan atrás con relación a soluciones comerciales existentes. Se adaptó un módulo de detección de anomalías para poder detectar amenazas que por su método de atacar son difíciles de identificar mediante reglas, como por ejemplo: gusanos y ataques de denegación de servicio. Se sometió al Suricata a pruebas de estrés para determinar su alcance y limitaciones. En los resultados se observó una relación entre procesamiento y

cantidad de memoria RAM utilizada con relación a la cantidad de tráfico y número de reglas cargadas en el IPS/IDS.

Finalmente se desarrolló una interfaz web para administración del IPS/IDS y se encapsuló la solución en un paquete rpm para una rápida instalación. Al final del documento se proponen mejoras futuras a la solución actual.

Indice General

Indice de Anexos

Indice de Figuras

Indice de Tablas

ABREVIATURAS

BPF	Berkeley Packet Filter
CPU	Central Processing Unit
DCERPC	Distributing Computing Environment Remote Procedure Call
DNS	Domain Name System
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
MVC	Model View Controller
NFQ	NetFilter Queue
NIC	Network Interface Card
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

CAPITULO 1

ANTECEDENTES Y OBJETIVOS

1.1 Sistemas de Detección y Prevención de Intrusos

1.1.1 Sistemas de Detección de Intrusos

El IDS cuyas siglas significa Intrusion Detection Systems (Sistema de Detección de Intrusiones), es un dispositivo de red que detecta accesos no autorizados y ataques a un computador o a una red basándose en patrones o firmas de amenazas conocidas, en políticas de seguridad configuradas e incluso por detección de tráfico anormal en la red.

El IDS es un dispositivo más de la red que se coloca en un lugar tal que pueda observar una copia del tráfico de la red que debe analizar y no necesariamente el tráfico original. Esta configuración suele recibir el nombre de **Modo Mirror**. Por su ubicación los IDS se clasifican en:

- **Network Based Intrusion Detection System (NIDS)**. Analiza el tráfico dirigido a toda una red.
- **Host Based Intrusion Detection System (HIDS)**.Analiza el tráfico dirigido un solo dispositivo de red (computadores, servidores).
- **Distributed Intrusion Detection System (DIDS)**.Una solo sistema que administra tanto los NIDS como los HIDS.

Cuando el IDS detecta un evento al que identifica como tráfico anormal o maligno, genera alertas. Dependiendo de la implementación, las alertas son registradas en: archivos (logs), base de datos, notificadas por correo o se pueden incluso enviar directamente a otras aplicaciones externas.

La respuesta de todo IDS es pasiva, es decir, únicamente alertan la presencia de un ataque, no lo detienen. Por esa razón los Sistemas de Detención de Intrusos son herramientas más útiles

para el analista de red, el mismo que se debe encargar de analizar y estudiar los eventos ocurridos en la red para realizar acciones que atenúen o eliminen futuras amenazas similares.

1.1.2 Sistemas de Prevención de Intrusos

Un sistema de prevención de intrusos (IPS) es un dispositivo que ejerce el control de acceso en una red informática para proteger a los sistemas computacionales de ataques. La tecnología de prevención de intrusos es considerada por algunos como una extensión de los Sistemas de Detección de Intrusos, pero en realidad es otro tipo de control de acceso, más cercano a las tecnologías cortafuegos. La mayoría de los IPS son implementaciones basadas en las versions IDS por lo que se suele asociar a estos dos dispositivos como uno solo. Más adelante se verán las diferencias primordiales entre estos dos componentes de red.

A diferencia de los IDS, los sistemas de prevención de intrusos no sólo alerta al administrador ante la detección de un ataque, sino que establece políticas de seguridad para proteger el equipo o la red de un ataque

1.1.3 Modos de detección de ataques de los IDS/IPS

Detección basada en Firmas

Una firma tiene la capacidad de reconocer una determinada cadena de bytes en cierto contexto, para activar una alerta. Por ejemplo, los ataques contra los servidores Web generalmente toman la forma de URLs. Por lo tanto se puede buscar utilizando un cierto patrón de cadenas que pueda identificar ataques al servidor Web. Sin embargo, como este tipo de detección funciona parecido a un antivirus, el administrador debe verificar que las firmas estén constantemente actualizadas.

Detección basada en políticas

En este tipo de detección, el IPS requiere que se declaren muy específicamente las políticas de seguridad. Por ejemplo determinar que hosts pueden tener comunicación con determinadas redes. El IPS reconoce el tráfico fuera del perfil permitido y lo descarta. Requiere de mayor trabajo por parte del administrador de red y es menos propenso a Falsos Veredictos, sin embargo no es tan completo o flexible como la detección por firmas.

Detección basada en anomalías.

Consiste en detectar condiciones anormales de la red. Para ello el dispositivo debe entrar primero en un modo de auto aprendizaje para detectar umbrales de normalidad y para que el administrador pueda afinar dado los Falsos Veredictos.

Este tipo de detección tiende a generar muchos falsos positivos, ya que es sumamente difícil determinar y medir una condición 'normal'. En este tipo de detección existen dos opciones:

- Detección Estadística de anormalidades: El IPS analiza el tráfico de red por un determinado periodo de tiempo y crea una línea base de comparación. Cuando el tráfico varía demasiado con respecto a la línea base de comportamiento se genera una alarma.
- Detección no Estadística de anormalidades: En este tipo de detección, es el administrador quien define el patrón 'normal' de tráfico. Sin embargo, debido a que con este enfoque no se realiza un análisis dinámico y real del uso de la red, es susceptible a generar muchos falsos positivos.

Detección Honey Pot

Aquí se utiliza un 'distractor'. Se asigna como honey pot un dispositivo que pueda lucir como atractivo para los atacantes. Los atacantes utilizan sus recursos para tratar de ganar acceso en el sistema y dejan intactos los verdaderos sistemas. Mediante esto, se puede monitorear los métodos utilizados por el atacante e incluso identificarlo, y de esa forma implementar políticas de seguridad acordes en nuestro sistemas de uso real **(1)**.

1.1.4 Diferencias entre IPS e IDS

Es común escuchar el concepto del IPS como “un IDS con capacidades de bloqueo”. Esta noción es correcta, mas no el concepto general de los IPS. Las diferencias entre estos dos dispositivos no se encuentran únicamente en su funcionalidad sino también en sus prioridades y características.

Un IDS es una herramienta de detección pasiva, no necesita estar en medio del tráfico para analizar los paquetes, le basta con recibir una copia del mismo. La caída del sistema resulta en un dolor de cabeza para el analista de red que perdió información valiosa durante el tiempo de fallo. El IPS, por otro lado, está en medio del

tráfico recibiendo todos los paquetes y decidiendo cual puede o no pasar, la caída del sistema es mucho más catastrófica que en los IDS dado que la red entera se puede caer junto al IPS si es que no existe una sistema de recuperación de fallos configurado en la solución.

En desempeño, los IPS necesitan estar en equipos más robustos debido a que de su velocidad de procesamiento de paquetes dependerá el throughput máximo al que estará limitado dicha red. En las redes comúnmente se dan ráfagas de tráfico llamadas “bursts”. Durante estos picos de tráfico, los IDS pueden almacenar en buffers de memoria los paquetes que no alcanza a procesar, para analizarlas cuando el tráfico se estabilice. Los IPS, no pueden almacenar en buffers tantos paquetes como los IDS debido a que esto aumentaría la latencia del dispositivo y por ende de la red. Durante ráfagas, los IPS tienen que descartar paquetes.

En cuanto a detección, los IDS pueden y suelen cometer errores; falsos positivos si es que alerta un evento que en realidad no es peligroso, y falsos negativos si es que no alerta un evento que en realidad sí es ataque. En los IDS, los falsos negativos son más perjudiciales, pues quitan información de amenazas al analista de

red. En cambio con los IPS, un falso positivo resulta en más que una alerta falsa, sino que termina en un bloqueo de paquetes inofensivos.

Los módulos de detección de anomalías son muy buenos para detectar actividades sospechosas que por alguna razón no pueden generalizarse en una regla. El problema con estos módulos es que pueden generar falsos positivos y como ya se explicó, los IPS no pueden cometer errores. Por esta razón este tipo de módulos son fuertemente recomendados únicamente para los motores IDS.

La siguiente tabla resume lo analizado en esta sección sobre las diferencias entre los Sistemas de Detección y Prevención de Intrusos.

	IPS	IDS
	Inline, Bloque Automático	**Mirror, Alertas para analistas**
Estabilidad	Caída del sistema es catastrófica para la red	Caída del sistema quita información al analista de red. No es algo crítico
Desempeño	Requiere mayor capacidad de procesamiento. Puede producir cuellos de botella.	La falta de procesamiento puede ser compensada con buffers de mucha memoria. Nunca producirá cuellos de botella.
Precisión de Falsos Positivos	Produce bloqueos de paquetes. Problema con aplicaciones.	Carga trabajo innecesaria para el analista en busca de falsas alarmas.
Precisión de Falsos Negativos	Paquetes maliciosos entran a la red. No es tan crítico como en el caso de los IDS.	Ataques resultan totalmente **invisibles** y pueden volver a ocurrir. Pérdida de información para el analista.

Tabla 1.1: Diferencias entre IDS e IPS

1.2 La actualidad de los IPS/IDS

Años atrás el Firewall de red se convirtió en un dispositivo necesario en toda red de datos, debido a los múltiples ataques de red en incremento.

Con el avance de las tecnologías las amenazas fueron evolucionando así como también los Firewalls de red hasta llegar a un punto que por sí solos no podían contener todos los tipos de amenazas existentes.

Hoy en día los Sistemas de Detección y Prevención de Intrusos están para cubrir las limitantes de los Firewall de red, por lo que su uso resulta hasta obligatorio en redes donde la información es un potencial activo.

De acuerdo al estudio de mercado realizado por Industrias Gartner **(3)** la cifra de mercado mundial de los IPS creció en un 5% en relación al 2009 quedando en $939 millones.

Las soluciones IDS/IPS también están evolucionando para adaptarse a las nuevas técnicas avanzadas de evasión que se aplican en las amenazas actuales **(2)**.

Técnicas como sistemas de detección de anomalías permiten detectar nuevo tipos de ataques basándose en el comportamiento anormal de la red.

Estos sistemas permiten prevenir ataques de tipo Denegación de Servicio, ataques de gusanos, e incluso cierto tipo de ataques de botnet.

También se están desarrollando nuevos algoritmos de detección de Malware basándose en su comportamiento en la red. Incluso se está aplicando la detección y prevención de intrusos para dispositivos móviles.

Las pequeñas y medianas empresas también se pueden beneficiar de la protección de los IDS/IPS sin tener que invertir enormes cantidades de dinero en soluciones propietarias.

Snort es un motor IDS/IPS de software libre cuya principal ventaja en la actualidad es su estabilidad ganada por sus años y años de desarrollo. Actualmente las comunidades han acusado al proyecto Snort de mantenerse estancado en su desarrollo.

Suricata por otro lado, tiene menos años de desarrollo, pero la falta de experiencia la compensa con una comunidad muy activa y en continuo trabajo de mejoras de la solución, con grandes planes a futuros para el proyecto.

1.3 Justificación y Objetivos

Tanto Snort como Suricata son motores IDS/IPS y no soluciones empresariales como tal, es decir que para que pueda ser adaptado a ser el

Sistema de Detección y Prevención de Intrusos de una empresa, es necesario integrarlo con módulos externos, como interfaces de monitoreo, colectores de alertas, reglas y firmas de seguridad, etcétera.

Actualmente existen muchas aplicaciones de software libre para ser integradas a estos sistemas, aunque la mayoría fueron diseñadas para Snort, pueden ser adaptadas a Suricata gracias a la compatibilidad con la que fue diseñada.

La continua actividad de la comunidad desarrolladora de Suricata, y las nuevas capacidades desarrolladas y en planes a futuro de este motor, han hecho de este proyecto open source el favorito de muchos desarrolladores en el área de la seguridad de información. El principal objetivo del presente documento es desarrollar una solución IDS/IPS para empresas usando a Suricata como motor de detección y prevención.

Los objetivos secundarios son:

- Describir el estado de arte del proyecto Suricata
- Someter al motor IDS/IPS a pruebas de rendimiento para definir sus limitantes
- Desarrollar un módulo de detección de anomalías y autoaprendizaje
- Desarrollar una interfaz de administración para la solución.

CAPITULO 2

DESCRIPCIÓN Y ESTADO ACTUAL DEL ID/IPS SURICATA

2.1 Descripción y Características de la Herramienta Suricata.

Suricata es el nombre de un proyecto de software libre para un motor Sistema de Detección y Prevención de Intrusos o de manera abreviada IDS/IPS; fue desarrollado por la comunidad de OISF (Open Infomation

Security Foundation).

El proyecto se empezó a mediados del 2007 y su versión beta estuvo disponible para descarga pública en enero del 2010 teniendo muy buena aceptación por tener características que cubrían algunas falencias en Snort, otra solución de IDS/IPS de código abierto que se venía usando hasta entonces.

El primero de julio, fue soltada la versión 1.0.0 estable haciendo completamente oficial el nacimiento de una solución que tal como sus desarrolladores aseguran “el propósito del proyecto no es el de reemplazar o emular a las herramientas actuales de IDS/IPS, si no aportar nuevas ideas y tecnologías para la próxima generación de Sistemas de Detección y Prevención de Intrusos”

Actualmente (Mayo, 2011) el equipo de OISF se encuentra desarrollando la versión Beta 2 de Suricata 1.1 disponible para su descarga en su página oficial. La última versión estable y disponible para su descarga es la 1.0.3

Es un proyecto relativamente nuevo, aún en continuo proceso de desarrollo, pero sus innovadoras características y su comunidad activa ha atraído cada vez más mirada de personas que trabajan en el área de la seguridad de información.

Características de la Herramienta Suricata

Multi-Threading: Esta característica es uno de los fuertes de suricata pues las soluciones software libre actuales de IDS/IPS y algunas de fabricantes, son uni-threaded. Multi-Threading consiste en procesar los paquetes en uno o más Hilos (Threads), y de esta manera se puede aprovechar el procesamiento multinúcleos de los actuales procesadores, haciendo que cada núcleo del procesador se encargue del procesamiento de uno o más hilos. Esto no es posible con soluciones como snort que son uni-threaded.

Estadísticas de Rendimiento: Este módulo se encarga de contar elementos de rendimiento como nuevas tramas/sec, duración, etc; y almacena esta información para presentarlos como estadísticas al administrador de alguna manera, ya sea vía logs, mensajes SNMP, vía web, etc.

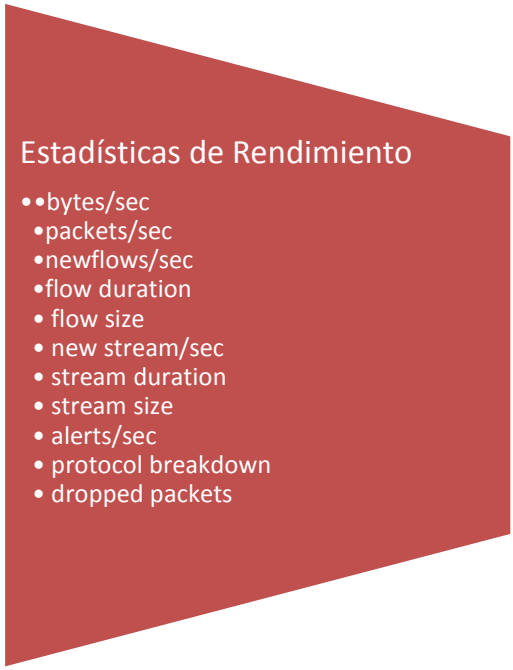


Figura 2.1: Datos que recolecta el módulo de Estadísticas de Rendimiento

Detección de Protocolos Automático: El motor de suricata tiene palabras claves para algunos protocolos como: IP, TCP, UDP, ICMP, HTTP, TLS, FTP y SMB. Esto quiere decir que se puede detectar una ocurrencia dentro de un stream de datos, sin importar el puerto en donde ocurre. Esta característica es importante para el control y detección de malware. Otros protocolos de capa 7 se encuentran aún en desarrollo.

Descompresión Gzip: Con la ayuda de la librería HTP es posible descomprimir un archivo en Gzip para examinarlo en busca de patrones de

ataque.

Independencia de la librería HTP: La librería HTP es un proyecto independiente a suricata e integrado efectivamente a suricata. Puede ser utilizado por otras aplicaciones como: proxies, filtros; y hasta el módulo mod_security de Apache.

Métodos de Entrada Estándar: Soporte para NFQueu, IPFRing y LibPcap standard para la captura de tráfico.

Unified2 Output: Soporte para métodos y herramientas de salida estándar Unified2. Este tipo de salida binario busca reducir carga al Suricata en cuanto a parseo de la información de salida, dejándole el trabajo a soluciones externas como Barnyard quien agarra los binarios, los interpreta y los almacena según donde configure el administrador.

Fast IP Matching: El motor de suricata puede usar automáticamente un preprocesador especial para validar más rápidos las reglas que hagan coincidencia únicamente de IP; por ejemplo, RBN, o las listas de ip de “EmergingThreats”

HTTP Log Module: Las peticiones de HTTP pueden retornar una respuesta con el formato de log de apache para monitoreo y registro de actividad.. Es posible usar Suricata únicamente como HTTP sniffer con este módulo.

IP Reputation: Consiste en compartir información de direcciones IP de mala reputación con otras organizaciones y soluciones de seguridad, para eliminar falsos positivos. Este módulo se encarga de colectar, almacenar, actualizar y distribuir el conocimiento de la reputación de direcciones IP. Esta calificación puede ser positiva o negativa y clasifica a las IP en categorías.

Funcionaría bajo una estructura “Hub and Spoke” donde en una base de datos central (Hub) se almacenaría y procesaría toda la información recolectada por los IDS para luego ser compilada y distribuida a la base de datos de los IDS clientes (Spokes).

La implementación técnica está dividida en tres componentes: La integración con el motor, el Hub, o base de datos central, que redistribuirá las reputaciones y el protocolo de comunicación entre Hubs y sensores (IDS’s)

Graphics Card Acceleration: usando CUDA y OpenCL se puede utilizar el poder de procesamiento de las tarjetas gráficas para acelerar el IDS y

quitarle carga al procesador principal para ganar rendimiento. Este módulo ya está incluído desde la versión 0.9.x de suricata pero aún sin los resultados esperados. Aún en desarrollo.

Windows Binaries: Suricata también puede correr en Windows, sobre versiones mayores a XP, aunque no es muy recomendable hacerlo pues es una solución inicialmente desarrollada para Linux que se caracteriza por ser más estable que Windows.

Flowint: Permite la captura, almacenamiento y comparación de datos en una variable global, es decir que permitirá comparar datos de paquetes provenientes de streams no relacionadas también. Se puede usar para un buen número de cosas útiles como contar ocurrencias, sumar o restar ocurrencias, activar una alarma al obtener un número x de ocurrencias, etc.

Módulo de capa de aplicación SSH: El interpretador SSH interpreta sesiones SSH y detiene la detección e inspección del flujo de datos después de que la parte de encriptación ha sido inicializada. Este módulo implementado en la versión estable 1.0.2, se concentra en reducir el número de paquetes que necesita inspección al igual que los módulos SSL y TSL **(7)**.

Más sobre el Multi-threading de Suricata

Como se ha mencionado, Suricata funciona a base de multi-hilos, usa múltiples núcleos del CPU para procesar paquetes de manera simultánea. Si está en un CPU con un solo núcleo los paquetes serán procesados uno por uno. Existen 4 módulos por hilo de CPU: Adquisición de paquete, decodificación de paquete, capa de flujo de datos, detecciones y salidas. El **módulo de adquisición de paquete** lee el paquete desde la red. El **módulo de decodificación** interpreta el paquete y se encarga de gestionar a qué stream pertenece qué paquete; el módulo de capa de flujo realiza 3 tareas: La primera, realiza lo que se conoce como 'tracking' o rastreo de flujo, que asegura que todos los pasos que se están siguiendo tienen una conexión de red correcta. La segunda: el tráfico de red TCP viene en paquetes, por lo tanto el motor de este módulo reconstruye el stream original. Finalmente la capa de aplicación será inspeccionada, tanto el flujo HTTP como DCERPC (Distributing Computing Environment Remote Procedure Call) será analizado. **Los hilos de detección** compararan firmas, pueden existir varios hilos de detección que pueden trabajar simultáneamente. En el **módulo de detecciones y salidas**, todas las alertas y eventos serán analizados.

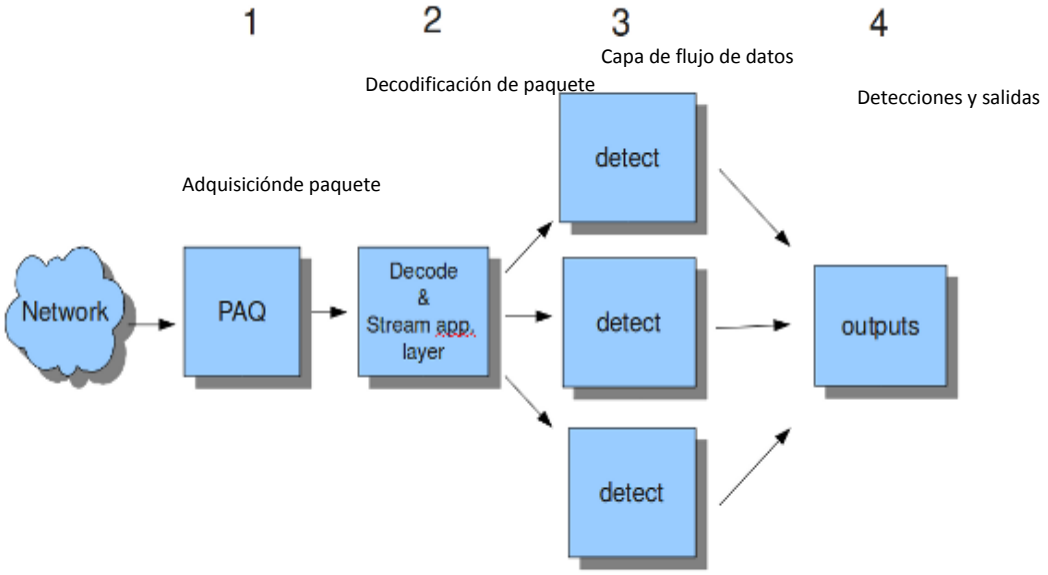


Figura 2.2: Proceso de los 4 módulos de Multi-hilos en Suricata.

En la figura 2.2, PAQ se refiere a la adquisición de paquetes. Decode se refiere a la decodificación de paquetes. Stream app. Layer realiza el seguimiento del flujo y reconstrucción. Detect, compara firmas y Output procesa todos los eventos y alertas.

La mayoría de computadoras tienen múltiples núcleos de CPU, por defecto el sistema operativo determina que núcleo trabaja en qué hilo. Cuando un núcleo está ocupado, otro será designado para trabajar en el hilo. Por lo tanto puede diferir a veces en que núcleo trabaja un hilo **(6)**.

En las versiones beta de Suricata, es posible asignar los procesadores que

se encargarán de trabajar los diferentes módulos. Además el multithreading funciona para todos los módulos y no sólo para el módulo detect, como en las versiones estables.

2.2 Comparación con otras soluciones IDS/IPS.

2.2.1 Soluciones Propietarias

Dado que estamos tratando con una solución reciente y gratuita, es de entender que las características de las soluciones propietarias van a superar notablemente a las de este IPS/IDS. Y eso también es lógico si tomamos en cuenta que obtener una solución propietaria requiere de una considerable inversión que, dependiendo de la robustez, desempeño y fabricante, va a estar entre los 5000 (para PYMES) a $200.000 (para service provider).

Gartner, una empresa consultora que realiza investigaciones en Tecnologías de Información, tiene un método de análisis y comparación de soluciones llamada “El Cuadrante Mágico de Gartner”.

Esta herramienta coloca a los fabricantes en una gráfica dividida en 4 cuadrantes en base a dos criterios:

- **Habilidad de Ejecución.**
- **Integridad de Visión.**

Según diferentes puntuaciones en estos dos criterios, los fabricantes caen en uno de los cuatro cuadrantes:

- **Líderes:** Alta habilidad de ejecución e integridad de visión
- **Competidores:** Alta habilidad de ejecución, baja integridad de visión.
- **Visionarios:** Baja habilidad de ejecución, alta integridad de visión.
- **Jugadores de Nicho:** Bajo en ambos criterios.

Entre las soluciones IPS/IDS propietarias destacan las de marcas conocidas como:

McAfee: Línea de productos IPS de McAfee. Tiene modelos que van desde los 100 Mbps hasta los 10 Gbps de througput. Se adaptan a otras soluciones empresariales de McAfee, como NAC, vulnerability management, ePolicy Orchestrator y host IPS.

CISCO: Disponibles dispositivos IPS dedicados como la serie IPS 42XX Sensor; por módulos adaptables a routers y switches como las

tarjetas IPS AIM y AIP-SSM; o soluciones de software como el CISCO IOS Intrusion Prevention System. Los dispositivos dedicados son más robustos y escalables soportando throughput de hasta 8 GBps y realizan prevención a nivel de red.

Sus IPS incluyen una característica llamada ***Risk Rating*** que permite ajustar alertas basadas en factores como la sensibilidad del activo que está siendo protegido. Esto provee un contexto para una mejor detección y bloqueo, sin embargo esta característica en manos de un administrador con poca experiencia podría terminar reduciendo la protección dada una mala configuración.

TippingPoint: Una línea de productos IDS/IPS de 3Com que luego en el 2010 fue adquirida por HP. Poseen varios modelos para cubrir diferentes necesidades de robustez y soportan hasta 8 Gbps de throughput.

Su fortaleza está en el módulo Core Controller, el cual permite hacer balanceo de carga entre múltiples dispositivos.

SourceFire: Línea de productos IPS de la empresa SourceFire, más conocidos por el desarrollos del IDS/IPS de software libre Snort y el

antivirus ClamAV. Tienen modelos que van desde los 5 Mbps hasta los 10 Gbps de througpput; ofrecen también soluciones virtuales de IPS para VMware y Xen.

Una de sus fortalezas es la flexibilidad otorgada para ver y hasta cierto punto modificar las reglas y firmas de seguridad, que en otras soluciones permanecen ocultas.

La figura 3.1 presenta el cuadrante mágico de Gartnier actualizado a diciembre del 2010 **(3)**.

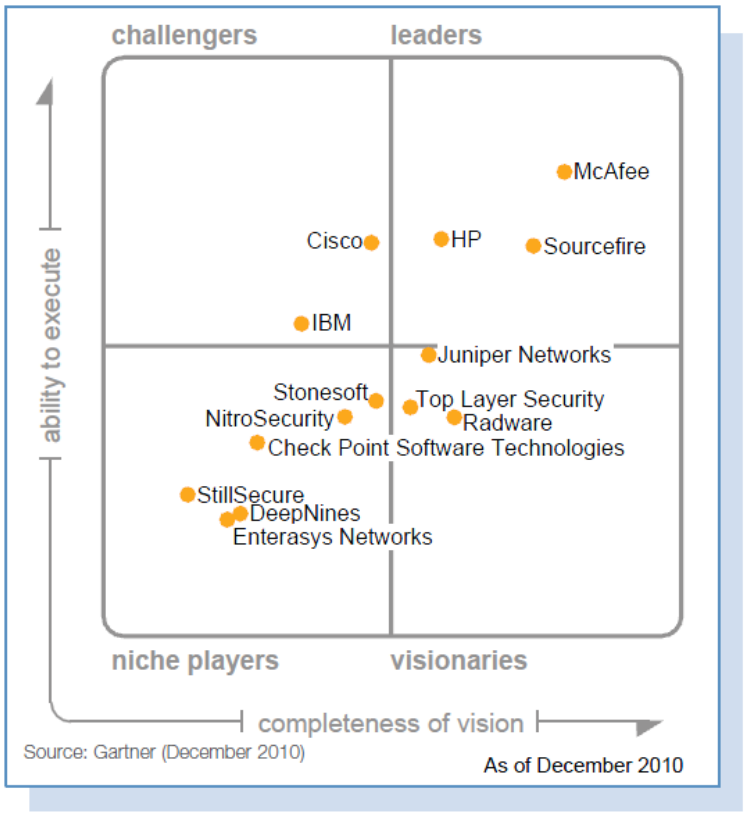


Figura 2.3 Cuadrante Mágico de Gartner para IPS's en el 2010

La Tabla 2.1 compara algunas características de suricata con otras soluciones IPS propietarias en general. La ventaja de Suricata al ser una solución de código abierto, es la flexibilidad de agregar nuevas funcionalidades al motor, según vayan siendo requeridas por las comunidades **(4)**.

Características	Soluciones Comerciales IDS/IPS	Suricata
Multi-Threading	x	Si
Soporte para IPV6	Cisco, IBM, Stonesoft	Si
IP Reputation	Cisco	Si
Detección Autamática de Protocolos	No	Si
Aceleración con GPU	No	Si
Variables Globales/Flowbit	No	Si
GeoIP	No	Si
Análisis Avanzado de HTTP	No	Si
HTTP Access Logging	No	Si
SMB Access Logging	No	Si
Anomaly Detection	Sí	No
Alta Disponibilidad	Si	No
GUI de Administración	Si	No
GRATIS	No	SI

Tabla 2.1 Comparación de Suricata con IPS/IDS's propietarias

2.2.2 Soluciones Propietarias

En cuanto a las soluciones de código abierto existen los que únicamente funcionan como Detección de Intrusos, siendo **Bro** uno de los más conocidos.

En cuanto a Detección y Prevención de intrusos, el más conocido es **SNORT**, un IDS/IPS desarrollado por Sourcefire; tiene más de 10 años de antigüedad y actualmente se encuentra cerca de la versión 2.9. Cuando la primero versión de suricata disponible fue lanzada, muchos se preguntaron si el proyecto snort moriría, pues últimamente no ha habido mucha innovación en este IPS. Sin embargo la entrada de suricata significó una "competencia" para snort, a la que la comunidad de sourcefire respondió con un mayor empeño en su proyecto.

Actualmente suricata posee características únicas que no se encuentran en snort (Tampoco en otros IPS) y de hecho, muchas veces se menciona a suricata como una actualización o mejora basada en snort, hecho que no es del todo cierto. La tabla 2.2 compara característica de suricata con Snort y Bro (como IDS) **(5)**.

Características	Bro	Snort	Suricata
Multi-Threading	No	x	Si
Soporte para IPV6	Si	Sí	Si
IP Reputation	Algo	No	Si
Detección Autamática de Protocolos	Si	No	Si
Aceleración con GPU	No	No	Si
Variables Globales/Flowbits	Si	No	Si
GeoIP	Si	No	Si
Análisis Avanzado de HTTP	Si	No	Si
HTTP Access Logging	Si	No	Si
SMB Access Logging	Si	No	Si

Tabla 2.2 Comparación de Suricata con IPS/IDS's de código abierto

2.3 Configuración General de Suricata

2.3.1 Requerimientos de Hardware y Software

Los requerimientos de Hardware para Suricata varían dependiendo de cuánto tráfico se espera que el sensor esté monitoreando, así como también de la cantidad de firmas de seguridad que serán cargadas.

A mayor throughput vamos a considerar tener un mejor procesador con mayor cantidad de núcleos. La memoria RAM por otro lado va a

tener que ser mayor conforme queramos aumentar el número de firmas de seguridad y también dependerá del throughput.

Si se ejecuta Suricata en modo IDS basta con una tarjeta de red para escuchar el tráfico de red y es recomendable una segunda interfaz de administración. En modo IPS por otro lado se necesitan mínimo 2 interfaces de red, las cuales estarán puenteadas (IPS es un dispositivo de capa 2) y es muy recomendable y casi necesario una interfaz más para administración.

Se recomienda también que se tengan las tarjetas NIC con capacidad de acuerdo al tráfico que existe en la red normalmente incluido el tiempo donde más actividad existe dentro de la red. Con esto las tarjetas NIC pueden operar acorde al tráfico de red a la cual el sensor está sometido.

En cuanto a los requerimientos de Software, sin importar la plataforma sobre la cual se va a instalar suricata, es necesario tener las siguientes herramientas instaladas:

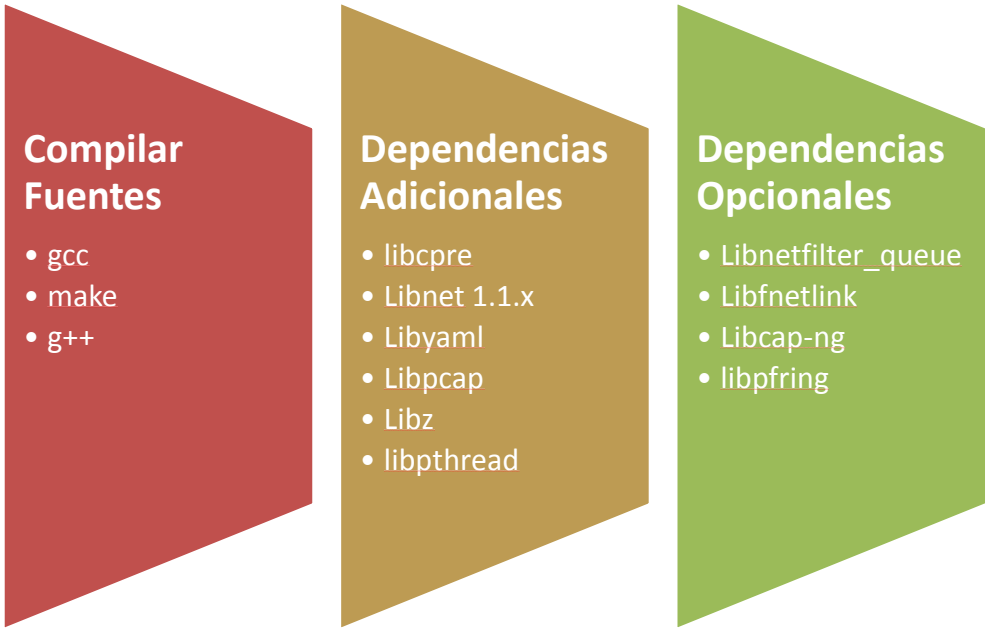


Figura 2.4: Dependencias y fuentes de Suricata

2.3.2 Archivo de Configuración Suricata.yaml

Suricata utiliza el formato YAML (Yet Another Markup Language) para la configuración general de su funcionamiento. La estructura de este tipo de formato se denota identando con espacios en blanco. Hay que tener mucho cuidado en no usar tabulación al trabajar con archivos YAML y también procurar que opciones del mismo nivel estén bajo la misma identación.

Entre las principales opciones de configuración para Suricata tenemos:

Max-pending packet

Con este ajuste se puede establecer el número de paquetes a procesar simultáneamente. Este campo debe ser escogido según el hardware con el que se dispone, tomando en cuenta que tiene una relación entre procesamiento y el uso de memoria RAM. Un mayor número de paquetes resulta en mayor procesamiento y mayor uso de memoria.

```
max-pending-packets: 2000
```

Action Order

Las reglas serán cargadas en el orden en el que aparecen en los archivos, pero serán procesadas en orden distinto, las firmas tienen prioridades diferentes. Las firmas más importantes serán escaneadas primero. El campo Action Order define esta prioridad

```
action-order:
 - pass
 - drop
 - reject
 - alert
```

Salida y Registro de eventos

Lo primero que hay que configurar es el directorio en donde se almacenarán los registros y salidas de Suricata. El valor por defecto es:

```
Default-log-dir: /var/log/suricata
```

Lo siguiente es configurar los tipo de salidas que se desean obtener de Suricata según lo requerimientos de la red. Hay salidas que pueden ser muy útiles en algunos casos como inútiles en otros.

Registro de alertas basado en líneas

Las alertas son almacenadas en un archivo de texto. Cada alerta ocupa una línea del archivo. Contiene una descripción de la alerta, el tiempo en el que fue activada y las direcciones IP comprometidas en

el evento. Útil cuando se desea obtener información rápida sobre eventos.

```
-fast:
    enabled: yes
    filename: fast.log
    append: yes/no
```

Salida de alertas para utilizar con Barnyard2 (unified2.alert)

Este registro es muy importante cuando se requiere enviar los eventos detectados por Suricata a una base de datos o aplicaciones externas. Barnyard2 es un colector que lee ficheros binarios de formato unified2 y los procesa para enviar el contenido (alertas) a donde se requiera **(6)**

```
- unified2-alert:
    enabled: yes
    filename: unified2.alert
    limit: 32
```

Registro de peticiones HTTP

Este registro mantiene un rastro de eventos que ocurren en el tráfico HTTP. Contiene HTTP Request, el nombre del host, URI y el agente usuario.

```
- http-log:
    enabled: yes
    filename: http.log
    append: yes/no
```

Salida a syslog

Con esta opción es posible mandar todas las alertas y eventos a syslog, el estándar para envió de registros en redes de datos.

```
- syslog:
  enabled: no
  facility: local5
. level: Info
```

Motor de Detección

El motor de detección construye grupos internos de firmas de seguridad. Suricata carga las firmas que serán comparadas con el tráfico. Ciertamente, muchas reglas no serán necesarias, por esta

razón todas las firmas serán divididas en grupos para optimizar el rendimiento del motor. Una distribución que contiene tantos grupos puede hacer uso de bastante memoria.

La cantidad de grupos determinarán el balance entre la memoria y el rendimiento. Una pequeña cantidad de grupos bajarán el rendimiento pero también usará poca memoria. Lo opuesto cuenta para cantidades grandes de grupos.

Si se dispone de un buen hardware y se espera procesar throughput mayores a 200 Mb, se recomienda configurar un perfil alto.

```
detect-engine:
  -profile: high
This is the default setting.
  - custom-values:
      toclient_src_groups: 2
      toclient_dst_groups: 2
      toclient_sp_groups: 2
      toclient_dp_groups: 3
      toserver_src_groups: 2
      toserver_dst_groups: 4
      toserver_sp_groups: 2
```

Afinidad de CPU

Muy útil cuando se dispone de más de un procesador en el Servidor donde se instalará Suricata. Esta opción permite asignar uno, varios,

o todos los procesadores a los diferentes hilos de ejecución (receive, detect, verdict, etc).

Cuando se selecciona más de un procesador para un hilo, se puede asignar tres diferentes modos: “Balanced”, para balancear el procesamiento entre los CPU asignados; y “Exclusive” para asignar específicamente el procesador que se encargará de un hilo.

```
cpu_affinity:
    - management_cpu_set:
        cpu: [ 4-7 ]
    - receive_cpu_set:
        cpu: [ all ]
    - decode_cpu_set:
        cpu: [ 0, 1 ]
        mode: "balanced“
    …..
```

Variables para Reglas

En las reglas que Suricata usará para determinar si los paquetes son peligrosos se utilizan variables definidas. Esto permite configurar las redes y direcciones IP a las que se quiere asignar reglas y a cuáles no. La implementación de variables no es algo nuevo, Snort usa la misma configuración y dado que Suricata es compatible con las reglas de Snort, también las soporta.

Es muy importante y necesario modificar estas variables adaptándolas a la red donde se encuentra Suricata.

```
Vars:
  address-groups:
    HOME_NET:
"[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    EXTERNAL_NET: any
    HTTP_SERVERS: "$HOME_NET"
    SMTP_SERVERS: "$HOME_NET"
    SQL_SERVERS: "$HOME_NET"
    DNS_SERVERS: "$HOME_NET"
    TELNET_SERVERS: "$HOME_NET"
```

2.3.3 Reglas y firmas de seguridad

Suricata por sí solo no tiene reglas con las cuales comparar los paquetes y discernir cuando hay ataques en la red.

Al igual que Snort, Suricata depende de reglas externas para su funcionamiento. Suricata es compatible con todas las reglas desarrolladas para Snort e incluso ha definido nuevos campos en las reglas que Snort no soporta.

Entre los desarrolladores que destacan por su continua actividad en el desarrollo y actualización de reglas y firmas de seguridad están: Emerging Threats y SourceFire (desarrolladores de Snort).

Existen sets de reglas gratuitas, así como un set de reglas pagadas. La diferencia está en el tiempo de actualización. El set de reglas

pagadas por lo general está al día en actualizaciones mientras que las gratuitas pueden tener hasta meses sin cambiar.

Para la solución se están usando las reglas gratuitas de Emerging Threats con licencia GPL, pues ellos están desarrollando reglas exclusivas para Suricata aprovechando sus implementaciones.

2.4 FODA de Suricata

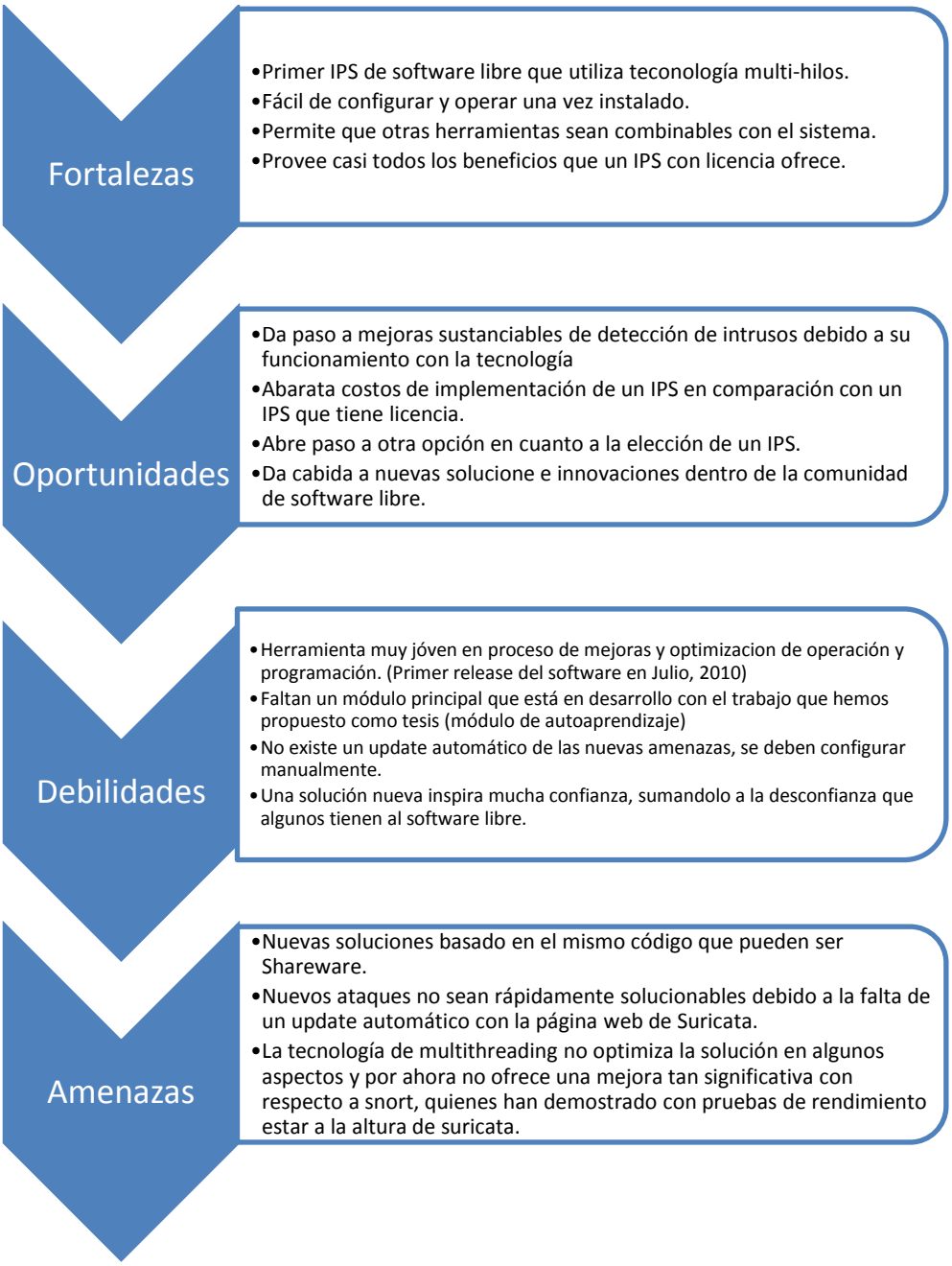


Tabla 2.3: FODA de Suricata

2.5 Futuras mejoras en próximas versiones de la herramienta suricata.

Actualmente Suricata se encuentra en la fase 2 de su proceso desarrollo. Los puntos a mejorar en esta etapa son:

- **Aceleración con CUDA GPU:** El uso de la unidad de procesamiento gráfico para acelerar el proceso de detección es un tema en el que se continúa trabajando y mejorando.

- **Reputación IP y DNS:** Existe la funcionalidad implementada en Suricata pero está por definirse y diseñar la arquitectura de nodos de distribución o hubs.

- **Salida por medio de Sockets en UNIX:** Permitirá que las alertas sean enviadas por sockets para que puedan ser utilizadas por algún otro proceso remoto.

- **GeoIP Keyword:** Usando bases de datos como Maxmind se puede obtener información geográfica de una dirección IP.

- **Estadísticas de Rendimiento:** Mostrar más información en la salida de

estadísticas (stats.log) con información de rendimiento.

- **Preprocesador de Anomalías.** Se busca implementar un preprocesador para detectar tráfico anormal de la red para prevenir ataques que no pueden ser definidos por reglas pero que presentan patrones en cuanto a comportamiento en red.

- **Salida a Mysql/Postgress/Sguil:** En vez de usar barnyard, mandar la información de alertas directamente a una base de datos. No es lo mejor para el rendimiento por lo que no será muy recomendable en ambientes de alto tráfico de red.

- **Actualización de Módulos sin reiniciar todo suricata.** Consiste en poder reiniciar ciertos módulos de Suricata como Rulesets, Salidas, Logs; sin necesidad de reiniciar todo el sistema y en lugar de eso se usaría un SIGHUP.
- **Embeber un sistema para prueba de reglas escritas por el administrador (7).**

CAPITULO 3

DISEÑO, E IMPLEMENTACIÓN DE LA SOLUCIÓN IDS/IPS EMPRESARIAL

3.1 Diseño de Interfaz Gráfica de Administración

Framework: Codeigniter

Para la interfaz que diseñamos, utilizamos CodeIgniter, desarrollado por EllisLab, un framework de PHP que nos brinda herramientas para desarrollar aplicaciones web dinámicas.

Nos provee varias librerías para las tareas que se necesitan comúnmente y además una interfaz simple con una estructura lógica para acceder a dichas librerías. Codeigniter utiliza la arquitectura MVC (Modelo Vista Controlador).

Arquitectura MVC (Modelo Vista Controlador)

Modelo Vista Controlador es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El patrón de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, dodne la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Descripción del patrón

- Modelo: Esta es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.

- Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Controlador: Este responde a eventos, usualmente acciones del usuario e invoca peticiones al modelo y probablemente a la vista.

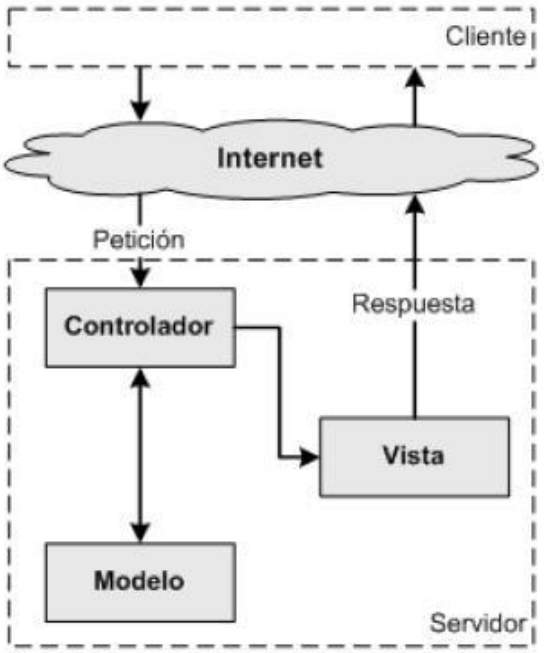


Figura 3.1: Diagrama de Arquitectura MVC.

Nosotros utilizamos una base de datos para gestionar datos de usuario, preferencias, rulesets, etc. Por lo tanto consideramos que es más conveniente utilizar esta arquitectura con el fin de promover una interfaz al usuario amigable, además para futuras mejoras de nuestro proyecto, que el desarrollador conozca de esta herramienta CodeIgniter y pueda desarrollar

en la misma, y que el acceso a los datos se encuentren siempre garantizados **(8)**.

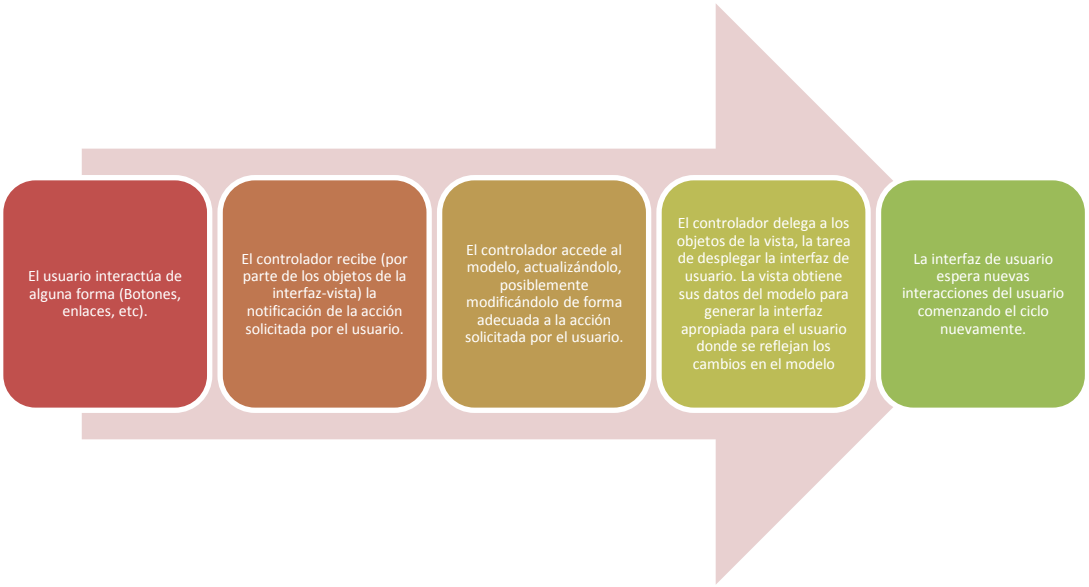


Figura 3.2: Proceso de la arquitectura Modelo Vista Controlador (MVC)

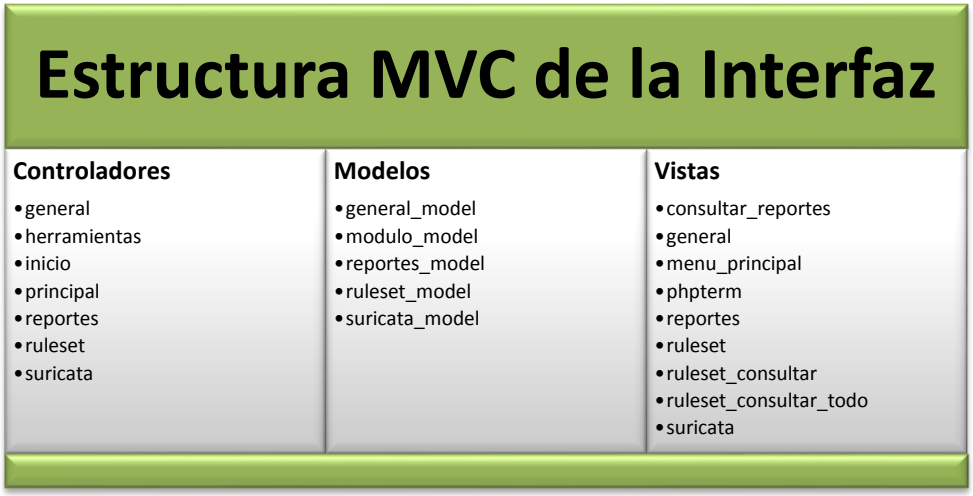


Figura 3.3 Estructura MVC de la Interfaz

3.1.1 Módulos de operación de la Interfaz

En la figura 3.4 los Controladores son propiamente los módulos de operación que conforman la interfaz. Es decir, todas las páginas que permiten ver y manipular opciones del motor.

Inicio: Es la página de autenticación. Sólo el administrador podrá ingresar a la interfaz de administración.

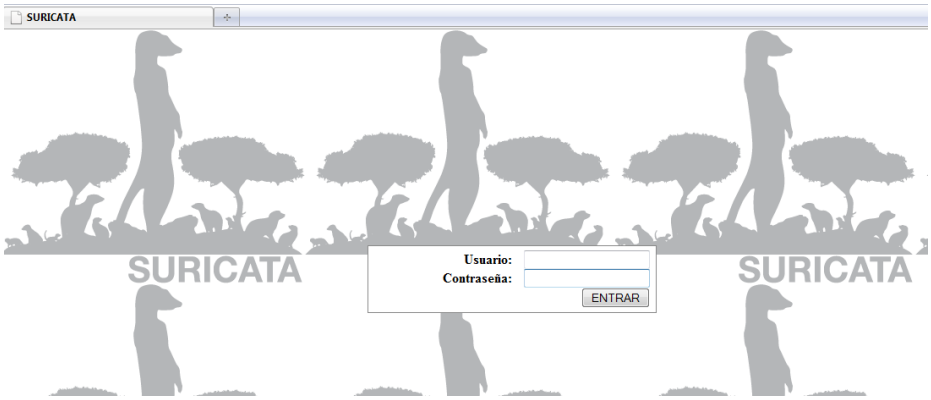


Figura: 3.4 Página Inicio. Autenticación

Una vez ingresados los datos correctos se podrá acceder a la interfaz, la misma que posee algunas pestañas de navegación.

Figura 3.5 Pestañas de Navegación de la Interfaz

General: Muestra información general de Hardware, interfaces de red y sistema operativo del servidor:

- Información de Procesador.
- Información de Memoria.
- Información de Red.
- Interfaces Puenteadas.
- Información de Sistema Operativo.

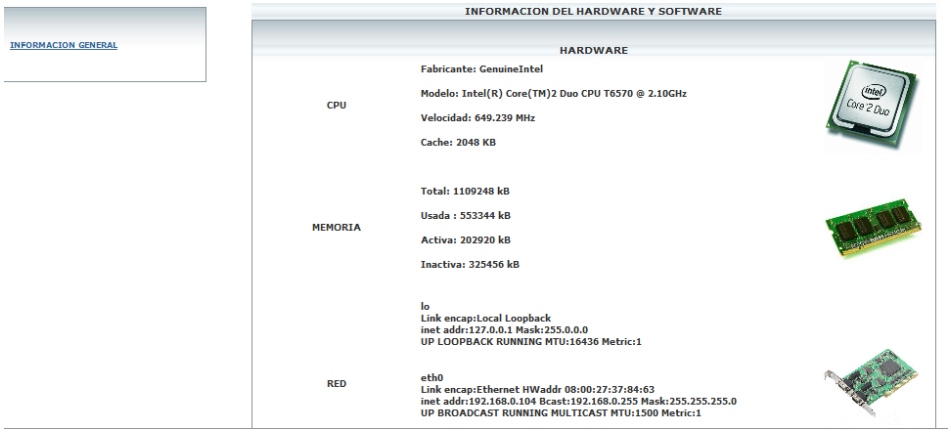


Figura 3.6 Página de Información General del Servidor

Suricata: Página para ver y configurar opciones del Sensor IDS/IPS.

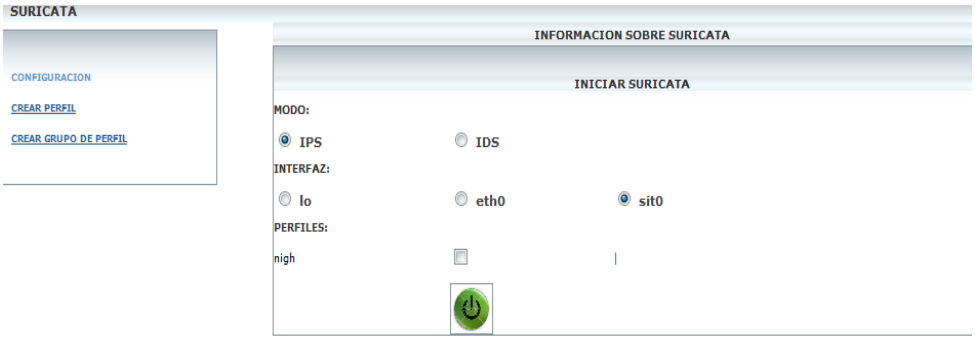


Figura 3.7 Página de Información y Configuración del Motor Suricata

Rulesets: Pagina para ver, habilitar y deshabilitar las reglas cargadas para un Perfil.

Internamente crea un directorio de reglas para cada perfil creado por el administrador. Cualquier cambio que se haga afectará únicamente a las reglas creadas para dicho perfil. Un perfil de reglas debe ser obligatoriamente escogido antes de iniciar Suricata, caso contrario tomará el perfil por defecto con reglas por defecto.

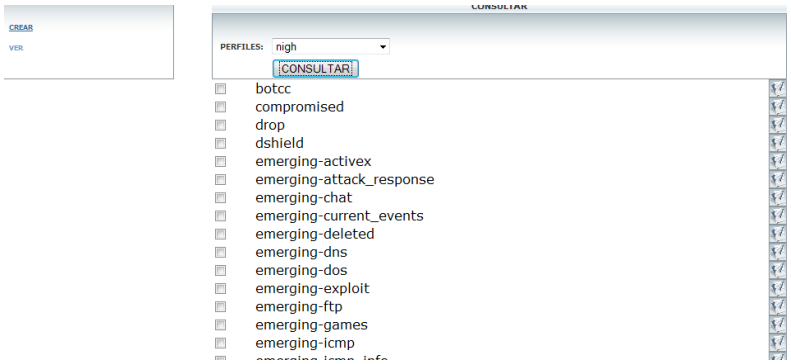


Figura 3.8 Página de configuración de Rulesets

Alertas: Página para ver las alertar generadas por el sensor. Se ha integra una interfaz existente llamada BASE, actualmente desarrollada para ver las alertas generadas por Snort pero que es 100% adaptable para Suricata.

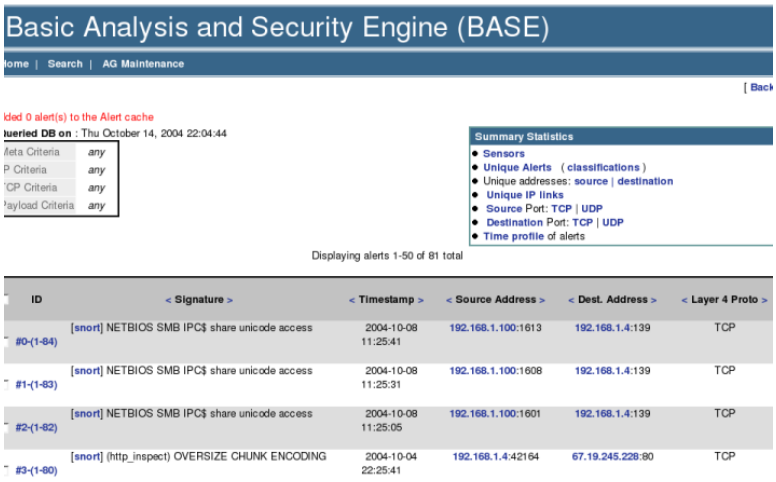


Figura 3.9 Página de monitoreo de Alertas usando BASE

Herramientas: Posee un Shell para configuración más específica del sensor.

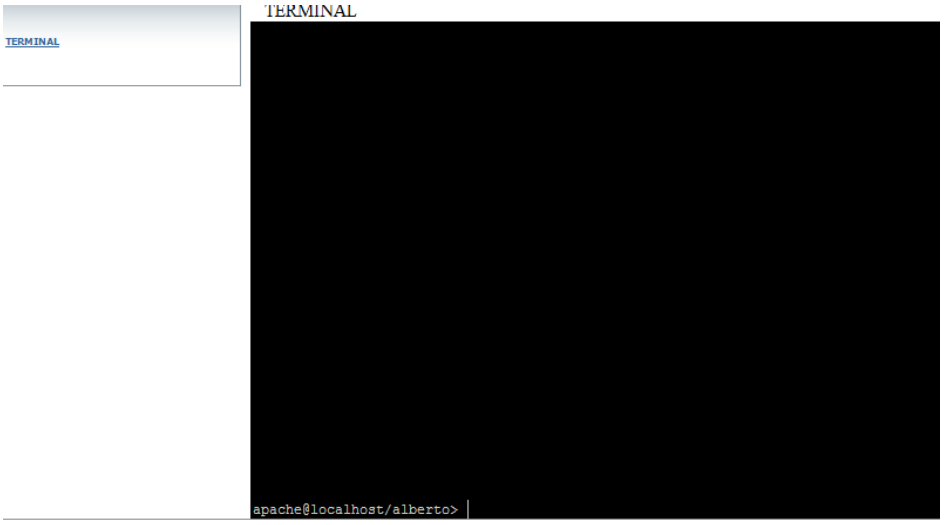


Figura 3.10 Terminal para configuración Interna del Sensor

Configuración de Variables: Para ingresar los valores correspondientes a las variables que se utilizan en las reglas de Emerging Threats.

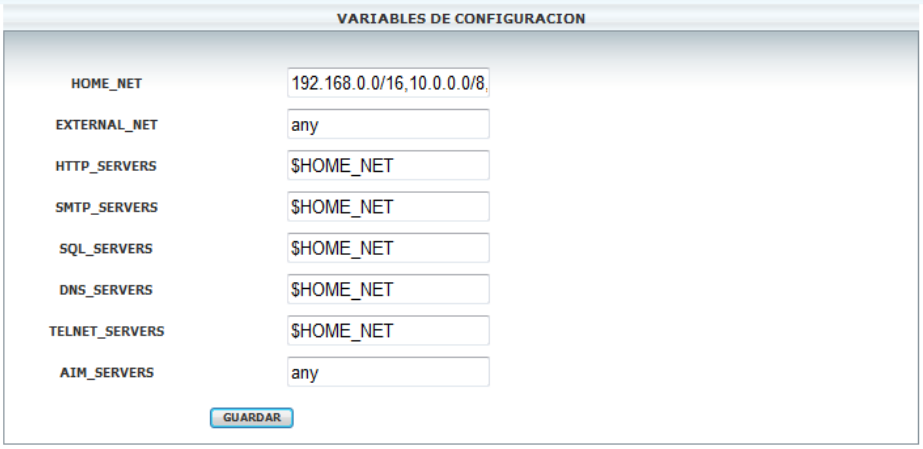


Figura 3.11: Terminal de variables de configuración de reglas

Graficas de monitoreo: Generadas cada minuto con información de estadísticas parseadas del archivo stats.log generado por Suricata. También hay información de rendimiento del equipo sacando la información de diversos comandos de Linux.

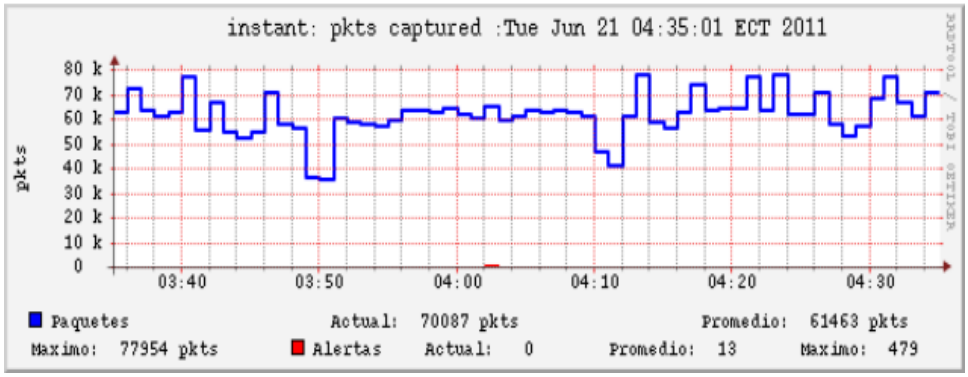


Figura 3.12 Monitoreo de paquetes capturados

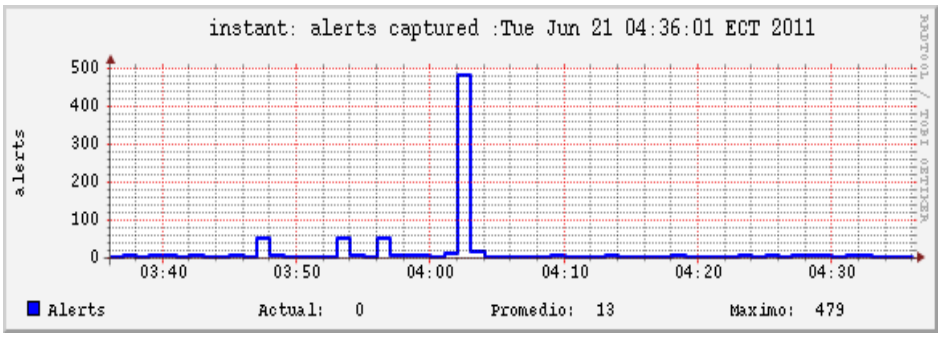


Figura 3.13 Monitoreo de alertas

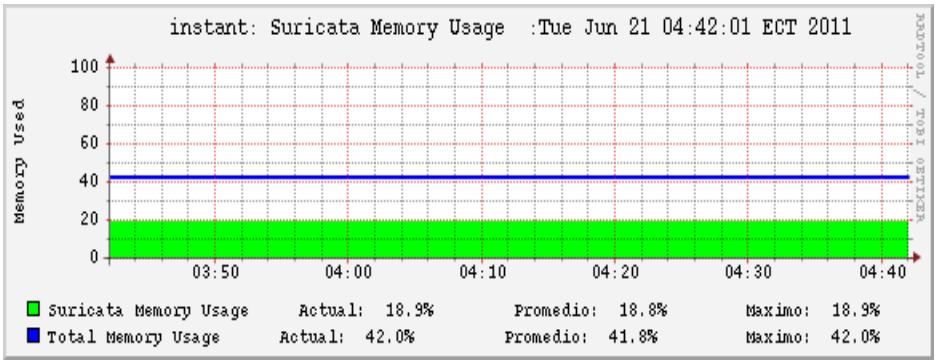


Figura 3.14 Monitoreo de uso de memoria

3.1.2 Diagrama de Diseño de la Base de Datos

Los ingresos de rulesets y las consultas que se realizan con nuestra interfaz dependen de una buena estructura de los datos. Un adecuado diseño de la base de datos es fundamental ya que en ella se almacenará tanto la información de permisos y roles que utilizará la interfaz, como también los eventos (ataques) que detecta el motor de Suricata.

El Apéndice B contiene la estructura de la base de datos con la descripción de las tablas y contenido.

3.2 Instalación del motor IDS/IPS Suricata para la solución Empresarial

Suricata puede ser instalado en varias plataformas. La documentación oficial de suricata ofrece instrucciones de instalación en varias plataformas, entre ellas: Linux (Centos, Ubuntu/Debian), Windows y MAC OS.

Pasos para instalar suricata en Centos 5:

1 Instalar dependencias:

```
yum install git
yum install libpcap libnet gcc automake autoconf make
yum install bridge-utils iptables iproute2
yum install libyaml libnfnetlink libnetfilter_queue

#Libcap-ng no se encuentra en repositories de arch por lo que tiene que #ser descargada y compilada manualmente
wget http://people.redhat.com/sgrubb/libcap-ng/libcap-ng-0.6.5.tar.gz
```

2 Descargar las fuentes de Suricata, o copiar las adjuntas (oisf.tar.gz) en cualquier directorio, se recomienda en ***/usr/local/src*** , descomprimir el archivo y ubicarse en la carpeta fuente:

```
cd /usr/local/src/
#Copiar en este directorio oisf.tar.gz y descomprimirlas con el comando:
tar -xvzf oisf.tar.gz
#Si se cuenta con conexión a internet las fuentes se pueden descargar:
git clone git://phalanx.openinfosecfoundation.org/oisf.git

cd oisf
```

3 Compilar los ficheros fuentes:

```
./autogen.sh
./configure –enable-nfqueue #enable-nfqueue para modo IPS
make
make install
```

Es necesaria la opción –enable-nfqueue en caso se quiera correr Suricata en modo IPS. Si no se agrega esa opción Suricata solo puede correr en modo IDS.

Se puede ejecutar: **./configure --help** para una lista de todas las opciones de compilación disponibles.

4 Crear usuario y grupo para Suricata:

```
useradd -M --shell /bin/false  --comment “Suricata IDP account” suricata
```

5 Crear directorios usados por Suricata y copiar en el directorio /etc/suricata los archivos de configuración localizados en el directorio fuente:

```
mkdir /etc/suricata          #Para archivos de configuración
mkdir /var/log/suricata      #Para logs de Suricata
cp /usr/local/src/oisf/suricata.yaml /etc/suricata/
```

6 Descargar y descomprimir las reglas de emerging threats en el directorio de suricata. Luego copiar del directorio de reglas archivos que usará el colector de alertar (barnyard2):

```
cd /etc/suricata/
wget http://www.emergingthreats.net/rules/emerging.rules.tar.gz
tar -xzvf emerging.rules
cp rules/classification.config .
cp rules/reference.config .
cp rules/sid-msg.map .
cp rules/gen-msg.map .
```

3.3 Configuración de la herramienta Suricata

Antes de poner en marcha la herramienta suricata en la solución Empresarial, hay que ajustar los parámetros de configuración, para el funcionamiento requerido del sistema, editando el archivo /etc/suricata/suricata.yaml copiado en el paso 5 de la instalación.

Los cambios realizados en el archivo de configuración, mientras suricata esté corriendo, no tendrán ningún efecto sobre su funcionamiento.

Algunas de las opciones configuradas en este archivo, pueden ser sobrescritas anteponiendo ciertos parámetros y argumentos al momento de ejecutar suricata, aunque no es algo muy común ni recomendable hacer. Entre las cosas que se deben configurar en el archivo suricata.yaml tenemos:

Como seteo inicial se debe editar principalmente los siguientes campos de suricata.yaml:

- Max-pending-packets
- Detect-engine
- threading
- outputs
- logging

- Rule-files
- Vars

El campo **max-pending-packets** aumenta el número de paquetes que pueden ser procesados simultáneamente y aumenta el buffer para las colas NFQ. Por defecto viene en 50 lo cual es un valor demasiado conservador y no servirá para troughput mayor a 100 megas

```
max-pending-packets: 2000
```

El campo **outputs** define los archivos salidas que presentará el Sensor. Es necesaria que esté habilitada **unified2-alert** (activada por defecto) y deshabilitar http-log ya que no se lo usará y consumirá recursos en vano.

```
- unified2-alert:
    enabled: yes

- http-log:
    enabled: no
```

El campo **detect-engine** configura el perfil de optimización de memoria al momento de agrupar y procesar reglas. Por defecto viene en médium. Se lo necesitará en high.

```
detect-engine:
  - profile: high
```

El campo **logging** permitirá habilitar los registros del sensor. Hay que habilitar el logging de tipo archivo.

```
- file:
    enabled: no
    filename: /var/log/suricata.log
```

El campo **threading** permite asignar CPUs por hilos de ejecución del motor Suricata. Por defecto cpu-affinity está deshabilitado así que hay que habilitarlo si el motor soportará tráfico mayor a los 100 Mbps y se desea tener control sobre los procesadores que se asignará a cada hilo.

Esta configuración dependerá del número de procesadores con el que se cuenta, a continuación se muestra lo ingresado para 8 procesadores.

Si algún procesador está consumiendo mucho más que otros, es necesario revisar esta configuración.

```
threading:
  set_cpu_affinity: yes

  cpu_affinity:
    - management_cpu_set:
        cpu: [ 0 ]  # include only these cpus in affinity settings
    - receive_cpu_set:
        cpu: [ "4-7" ]  # include only these cpus in affinity
settings
        mode: "exclusive"
        prio:
         default:"high"
    - decode_cpu_set:
        cpu: [ 0, 1 ]
        mode: "balanced"
    - stream_cpu_set:
        cpu: [ "0-1" ]
    - detect_cpu_set:
        cpu: [ "0-3" ]
        mode: "exclusive" # run detect threads in these cpus
        prio:
          low: [ 0 ]
          medium: [ "1-2" ]
          high: [ 3 ]
          default: "medium"
    - verdict_cpu_set:
        cpu: [ "4-7" ]
        mode: "exclusive"
        prio:
          default: "medium"
    - reject_cpu_set:
        cpu: [ 0 ]
        prio:
          default: "low"
    - output_cpu_set:
        cpu: [ "0-3" ]
        prio:
           default: "medium"
```

El campo **rule-files** indica las reglas que se cargarán en memoria al inicio de Suricata, y por ende contra las que se compararán los paquetes en busca de amenazas.

```
 default-rule-path: /etc/suricata/rules/
rule-files:
 - botcc.rules
 - emerging-attack_response.rules
 - emerging-dos.rules
 - emerging-exploit.rules
 - emerging-malware.rules
 - emerging-p2p.rules
 - emerging-scan.rules
 - emerging-voip.rules
 - emerging-web_client.rules
 - emerging-web_server.rules
  . . .
```

El campo **vars** contiene variables que se usan en las reglas de emerging threats. Por defecto HOME_NET viene con todo el rango de direcciones IP privadas. Es necesario editar estas variables para que se ajusten al entorno donde estará ubicado el Sensor.

```
vars:
  address-groups:
    HOME_NET: "[186.3.0.0/16,186.5.0.0/16,201.218.0.0/16,… ,172.16.0.0/12]"
    EXTERNAL_NET: any
    HTTP_SERVERS: "$HOME_NET"
    DNS_SERVERS: "[200.93.192.148,200.93.192.161]"
```

Ejecutando Suricata

Suricata, nativamente soporta dos modos de operación:

Intrusion Detection System (IDS).- Suricata captura paquetes con la librería pcap y alertará cuando un paquete active una de las firmas de seguridad del sensor. Se debe seleccionar la interfaz que escuchará todo el tráfico de la red en modo promiscuo.

Para ejecutar Suricata en modo IDS el comando es:

Suricata –c /etc/suricata/suricata.yaml –i interfaz

Intrusion Prevention System (IPS).-Suricata actúa en modo INLINE, es decir que todo el tráfico de red pasa obligatoriamente a través de él. Si algún paquete activa alguna firma de seguridad, el sensor puede descartarlo previniendo el ataque a la red. Para poder funcionar en modo IPS, suricata

tuvo que haber sido compilado habilitando NFQUEUE (Ver instalación de Suricata)

NFQUEUE permite encolar paquetes en diferentes grupos definidos por un Identificador de cola. Es neceario crear una regla en iptables que mande a alguna cola los paquetes entrantes.

El comando para ejecutar suricata en modo IPS es:

```
Suricata –c /etc/suricata/suricata.yaml –q Id_cola
```

Desde la versión 1.0.1 es posible especificar más de un id_cola. Suricata asignará threats individuales para cada cola.

```
Suricata –c /etc/suricata/suricata.yaml –qId_cola1 –qId_cola2 –qId_colaN
```

En las reglas de iptables se debe enviar el tráfico deseado a la o las colas que Suricata lee.

Para una cola:

```
Iptables –A FORWARD –i interfaz –j NFQUEUE
```

Para cuatro colas:

```
iptables -A FORWARD  -m statistic --mode nth --every 4 -j NFQUEUE --queue-num 1
iptables -A FORWARD  -m statistic --mode nth --every 3 -j NFQUEUE --queue-num 4
iptables -A FORWARD  -m statistic --mode nth --every 2 -j NFQUEUE --queue-num 3
iptables -A FORWARD  -j NFQUEUE --queue-num 2
```

Finalmente se puentean las interfaces por las cuales pasará el tráfico de red.

```
brctl addbr br0
brctl addif br0 eth2
brctl addif br0 eth1

ip li set br0 up
ip li set eth2 up
ip li set eth1 up
```

3.4 Módulo de Recuperación de Fallo

Este módulo es importante para la ejecución de la Solución en modo IPS. En modo inline el tráfico pasa a través de dos interfaces de red puenteadas. Mediante reglas de iptables se redirige el tráfico deseado a Suricata para su análisis, sin estas reglas puestas, el tráfico pasaría directamente de una interfaz a otra sin pasar por Suricata. Cuando esto sucede decimos que Suricata está cortocircuitado.

Basándonos en este principio, se ha elaborado un programa que continuamente envía paquetes ligeros a través de las colas donde Suricata recibe paquetes.

Si no se detecta estos paquetes generados por el programa en un intervalo de tiempo definido, se cortocircuita Suricata hasta que este se recupere del congestionamiento o cualquier situación que impidió que se lean los paquetes de monitoreo.

La siguiente figura muestra el algoritmo usado para la recuperación de fallo.

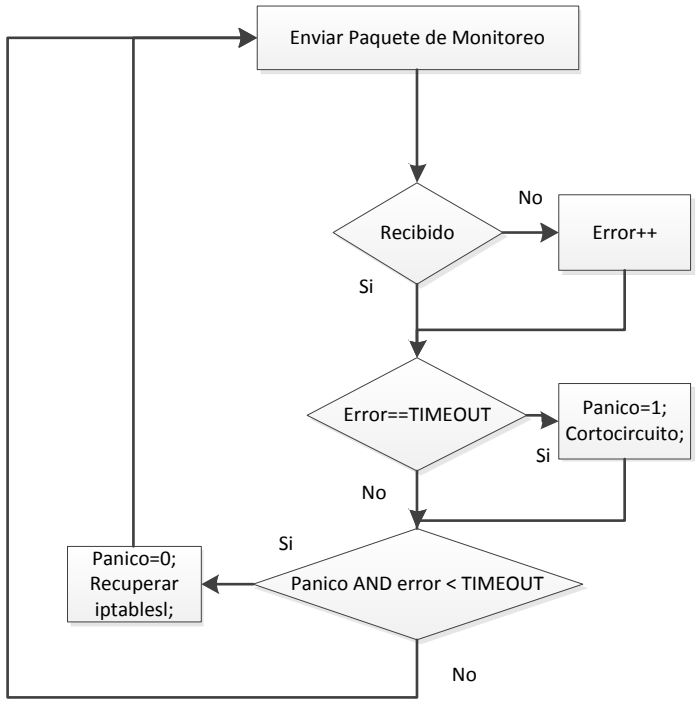


Figura 3.15 Diagrama de estados del módulo de recuperación de fallos

3.5 Módulo Colector de Alertas

El modulo colector de Alertas se encarga de recoger los archivos de salida estándar Unified2 de Suricata, donde se encuentran los eventos (alertas) registrados por el motor, y los interpreta según su configuración. Lo más común es guardar los eventos en una Base de datos. Eso es lo que se hará a continuación.

Instalación de Colector de Alertas (Barnyard2)

4 Instalar dependencias

```
pacman -S libmysqlclient
```

5 Copiar las fuentes de barnyard2 adjuntas en **/usr/local/src** y compilarlas

```
./configure --bindir=/usr/bin --sysconfdir=/etc/barnyard2 --with-mysql-libraries=/usr/lib  --with-mysql-includes=/usr/include/
make
make install
```

6 Agregar en el archivo **/etc/rc.local** el comando de ejecución de barnyard para que corra siempre al inicio.

```
barnyard2 -c /etc/barnyard2/barnyard2.conf -d /var/log/suricata/ -f unified2.alert -w barnyard2.waldo -D
```

Configuración de Colector de Alertas (Barnyard2)

1. Editar el archivo **/etc/barnyard/barnyard2.conf**

```
config reference_file:      /etc/suricata/reference.config
config classification_file: /etc/suricata/classification.config
config gen_file:            /etc/suricata/gen-msg.map
sid_file:           /etc/suricata/sid-msg.map
…..
config logdir: /var/log/
……
output database: alert, mysql, user={usuario} password={clave} dbname=sar host={db_server_ip}
```

En la última línea hay que llenar los valores entre llaves con los correctos según la configuración de la base de datos a la que se conecta barnyard.

3.6 Instalador de la Solución Empresarial.

Para crear un instalador para nuestra Solución Empresarial decidimos convertir a Suricata en paquete (RPM) e integrándolo a una distribución Linux CentOS, de esta manera cuando el usuario instale CentOS, dentro de los paquetes se le instalará, estará Suricata con Libpcap incluido.

Para realizar esto seguimos los siguientes pasos: Creamos cinco subdirectorios para el RPM Suricata:

- BUILD: La carpeta donde se va a compilar el software.
- RPMS: Contiene el binario RPM donde rpmbuild es construido.

- SOURCES: Aquí van los códigos fuentes de Suricata.
- SPECS: El script que se encargará de construir el paquete RPM.
- SRPMS: Contiene el archivo fuente del RPM construido durante el proceso.

Se empaquetó a Suricata en un tarball detallando la versión de la aplicación para diferenciarla de futuras versiones de Suricata. En este caso se lo llamo package-version.tar.gz.

Después creamos un archivo .spec, específicamente (nombre del archivo spec). El script sirve para dar instrucciones de compilación e instalación de Suricata una vez ya empaquetada la solución. El archivo esta especificado en el anexo 1.

Una vez terminado el archivo .spec construimos el paquete RPM ejecutando la siguiente instrucción:

$ rpmbuild –v –bb –clean SPECS/archivospec.spec

La instrucción rpmbuild hace lo siguiente:

- Lee y analiza el archivo suricata-1.1.spec

- Ejecuta la sección %prep para desempaquetar el código fuente en un directorio temporal, dicho directorio temporal es BUILD.
- Ejecuta la sección %build para compilar el código
- Ejecuta la sección %install para instalar el código en los directorios en la máquina build.
- Lee la lista de archivos de la sección %files, los agrupa y crea el binario RPM (y los archivos RPM elegidos).

Con esto tenemos empaquetado correctamente para la instalación a Suricata [9]. Como Suricata necesita de Libcap, se crea un repositorio Yum que contiene tanto el RPM de Suricata como el RPM de Libcap de la siguiente manera:

Instalamos primero ‘createrepo’ con el comando ‘yuminstallcreaterepo’ con este RPM podremos crear paquetes de repositorios YUM. Una vez compilado e instalado createrepo se ejecuta el comando con un argumento, dicho argumento representara la carpeta donde se generará la data para el repositorio. En este caso se ejecutó la instrucción:

```
createrepo /home/Suricata
```

Esto nos creó 4 archivos en el directorio que hemos creado con la instrucción ‘createrepo’. Luego de esto agregamos un archivo .repo en el directorio /etc/yum.repos.d/ llamado Suricata.repo, que es un script que contiene lo siguiente:

```
[Suricata_Repo]
name = Suricata Repo
baseurl = file:///home/Suricata
```

Con este script, cuando hagamos yuminstallSuricata, se crearán los paquetes RPM tanto de Suricata como el de Libcap **(10)(11)**.

CAPITULO 4

DISEÑO E INTEGRACIÓN DEL MÓDULO DE DETECCIÓN DE ANOMALÍAS

4.1 Antecedentes de módulos de detección de anomalías

Los IDS/IPS han sido estudiados de manera extensa en los últimos 20 años, sin embargo a finales de los años 90 estas técnicas detectaban de manera limitada los tipos de ataque que invadían en la red y reportaban un número alto de ataques falsos.

Por este motivo se introdujo un método de monitoreo que consiste en detectar anomalías en la red. Denning **(12)** formalizó el concepto de detección de anomalías asumiendo que las violaciones de seguridad pueden ser detectados inspeccionando aquellas direcciones IP que solicitaban utilizar recursos en dimensiones anormales, como resultado, estas técnicas de detección de anomalías buscan establecer actividades dentro del tráfico “normal” de una red mediante métricas, y en caso de ocurrir un ataque, dicho comportamiento de tráfico será completamente distinto al que el dispositivo ya ha estudiado. Pero para conseguir esto, se requiere de algoritmos para detectar tráfico anómalo que más adelante serán explicados.

De acuerdo a Axelsson **(13)**, la detección de anomalías se lo considera como auto aprendizaje debido a que estos sistemas forman una opinión acerca del comportamiento normal del tráfico. Se pueden aplicar varias técnicas de detección de anomalías, tales como información estadística de comportamiento, entrenamiento del motor de detección para saber cómo es el tráfico de red normalmente, entre otros.

El módulo de detección de anomalías de por sí solo detecta mas no alerta, queda a discreción del administrador de redes de observar y juzgar de acuerdo a los reportes que presenta el módulo, si es o no un ataque de red.

Por eso debe combinarse con los IDS/IPS para alertar y combatir de manera automática y autónoma ataques tales como DDoS, de gusano, scanneo de red o de botnets.

Un ejemplo de los primeros módulos de detección de anomalías fue propuesto por Wei Lu y Hengjian Tong **(14)** utilizando dos detectores de anomalías (CUSUM y EM) con el IDS/IPS Snort. En la figura 4.1 se observa la arquitectura del framework que ellos usaron para su IDS/IPS "híbrido".

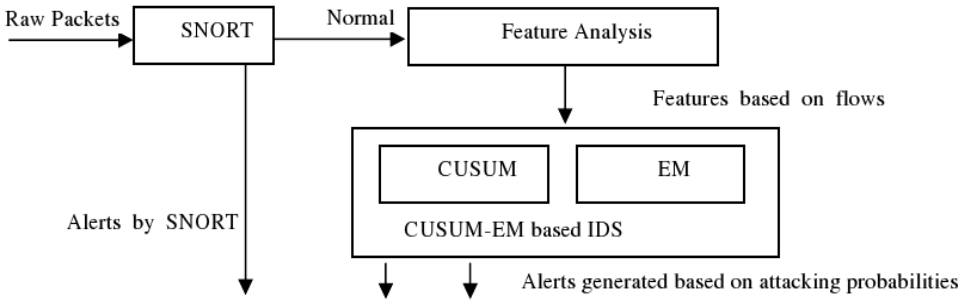


Figura 4.1 Arquitectura general del módulo de detección de anomalías.

Este módulo fue puesto en acción durante 5 semanas estableciendo un historial de comportamiento dentro de la red a la que fue puesta. Una vez que el framework aprendió del comportamiento de la red, detectó aproximadamente 13 tipos de ataques que Snort no podía detectar debido al conocimiento limitado de nuevos ataques que tuvo Snort y que los dos módulos de anomaly detection lograron detectar.

4.2 Herramienta Ourmon

Ourmon es un sistema de detección de anomalías de código abierto creado por el Ing. James Binkley de la universidad de Portland, Oregon. Está basado en la recolección de paquetes en modo promiscuo utilizando las interfaces Ethernet. Un módulo recoge los paquetes y los envía internamente en tuplas definidas a un módulo gráfico, que muestra de manera gráfica el flujo de red en base a cada dirección IP que está en dicha red. Este módulo puede estar o no en el mismo host que Ourmon.

Ourmon analiza los datos utilizando varias instancias de BPFs (Berkeley Packet Filter) y los muestra a través de herramientas gráficas tales como RRDTOOL, histogramas y reportes ASCII. Los datos son producidos cada 30 segundos y se muestran reportes en cada hora de direcciones IP que están circulando por la red.

Esta herramienta es configurable y provee gráficos, registros y reportes respecto a varios tipos de flujos, incluyendo los flujos de tráfico convencionales como flujo IP, SYNS, Ports, Errores ICMP y UDP entre otros.

Ourmon además utiliza varios algoritmos para detectar posibles ataques sospechosos, debido a la recolección de datos que es almacenado en tuplas

y está basado en las direcciones IP host donde se puede proveer información de ataques coordinados y gusanos **(15)**.

- Algunas cosas que Ourmon puede hacer son las siguientes:
- Monitorear flujos TCP y UDP
- Ayuda a medir el tráfico de manera estadística
- Atrapa servidores de correo en modo relay.
- Atrapa “botnets”
- Descubre infecciones con malware “zero-day”
- Descubre ataques desde adentro o desde afuera.
- Observa que protocolos están ocupando el mayor ancho de banda.

4.3 Algoritmo de detección de anomalías usado por Ourmon.

4.3.1 Anomalías en tráfico TCP.

El algoritmo de de anomalías en tráfico TCP ordena los paquetes por tuplas SYN. Estas tuplas consisten en una agrupación de atributos básicos con una dirección IP que es la que será monitoreada, la tupla SYN tiene la siguiente forma:

(IP Source address, SYNS, SYNACKS, FINSSENT, FINSBACK, RESETS, ICMP ERRORS, PKTSSENT, PKTSBACK, port signature data)

La clave lógica en esta tupla es que el IP Source address, SYNS, FINS y RESETS sean contadores de los paquetes TCP.

- **SYNS** son contadores de paquetes SYN enviados desde el IP fuente.
- **SYNACKS** es un subset de aquellos SYN enviados con la bandera ACK.
- Los **FINS** son una bandera que notifica que no hay más datos que mandar desde la fuente.
- **RESETS** son contados cuando son enviados de regreso a la IP fuente.
- **ICMP Errors** se refiere a ciertos errores de ICMP que son inalcanzables.
- **PKTSSENT** cuenta los paquetes que han sido enviados por la IP fuente.
- **PKTSBACK** son los paquetes que retornados a la fuente IP.

Existen dos “pesos” asociados a la tupla SYN, la cual Ourmon la llama “peso de trabajo” y “peso de gusano”. El peso de trabajo es computado por IP fuente con la siguiente fórmula:

$$(S_S + F_s + R_r)/T_{sr}$$

Siendo la terminología la siguiente:

S = SYN s = sender

F = FINS r = receiver

R = RESETS

T = TCP

El resultado de esta expresión está dado en porcentaje. La idea es controlar los paquetes para ser usados de una manera anómala y dividir ese contador por el total de paquetes TCP. Si da 100%, es una mala señal e implica que una anomalía de un tipo se está dando. Dicho valor se asocia con un posible scanner o un gusano. En otro caso si un intercambio de paquetes ordinarios se da, el peso será mucho menor, inclusive puede llegar al 0%.

Para calcular el peso de gusano, se aplica la siguiente fórmula:

$$S_s - F_r > C$$

Donde se extrae el FINS retornado por los SYNS enviados y solo almacena la tupla si C (que es una constante) es mucho mayor que

una constante configurable manualmente. Ourmon pone como default la constante C como 20 de manera intuitiva justificando que dicha constante debe ser lo suficientemente grande que cualquier IP fuente esté generando más SYNS que FINS. Con esto habrá IP fuentes que serán mayores al peso de la constante C.

Con estos datos, tanto del peso de trabajo como del peso de un gusano, logramos construir un histograma para que la solución empresarial propuesta en esta tesina de seminario, aprenda basado en detección de anomalías cuando se trata de un ataque gusano o de otro sospechoso activando los rulesets que tiene configurados de acuerdo a la red que se está monitoreando **(16)**.

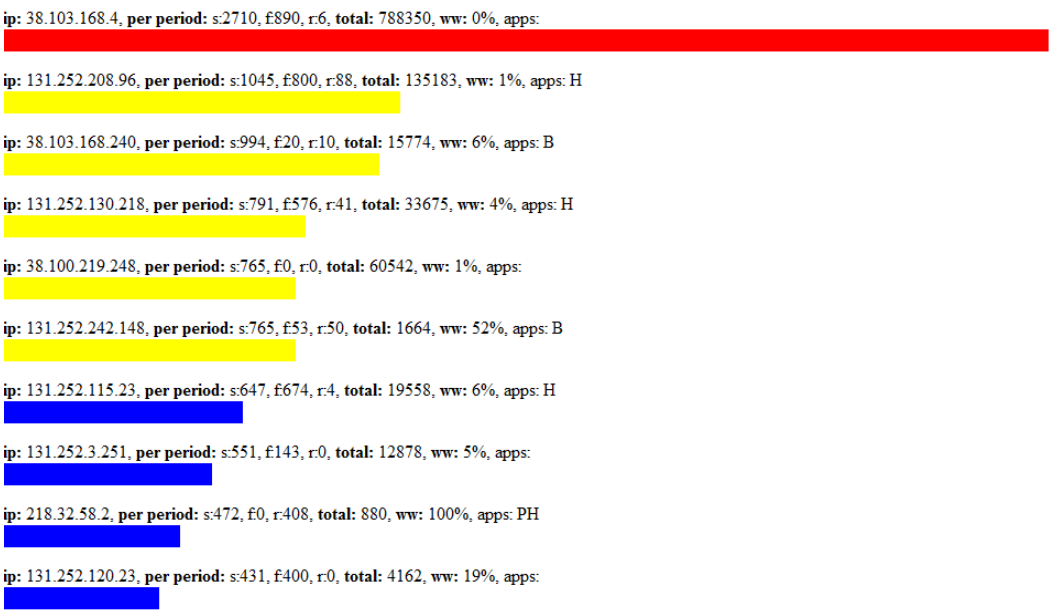


Figura 4.2: Lista de IPs que entran al campus.

Se observa en la figura 4.2 las últimas direcciones IP que han estado dentro de la red del campus en Portland, Oregon, se puede observar que la IP 218.32.58.2 presenta un WW (peso de trabajo) de 100% debido a que había 0 FINS por lo tanto se mandaba paquetes sin información que representa un posible escaneo de la red.

Además de un histograma para la detección de gusanos usando el algoritmo que usa Ourmon, también agregamos un algoritmo para detectar botnets dentro de la misma red. Debido a que la principal vía de infección es mediante IRC: Internet Relay Chat, protocolo de comunicación en tiempo real basado en texto, donde el hacker se comunica con las computadoras infectadas **(16)**.

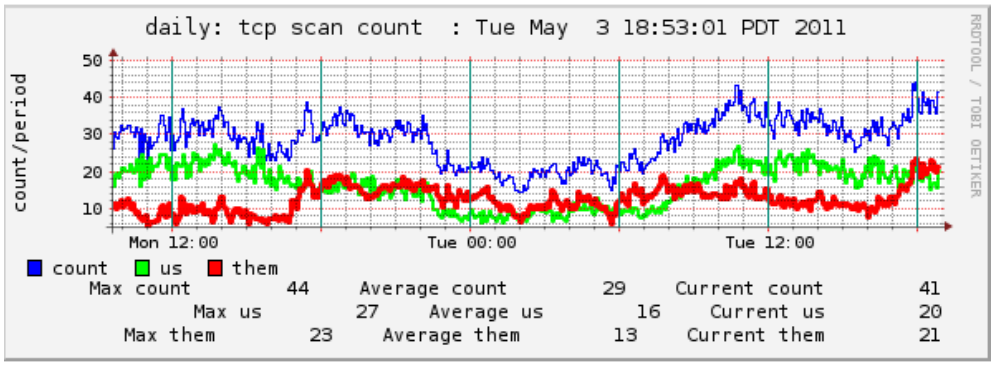


Figura 4.3 Grafo de detección de gusanos utilizando Ourmon.

En la figura 4.3, Ourmon está detectando en vivo el flujo de red, donde hay 3 datos: La línea azul suma la cantidad de tráfico TCP que está

circulando en la red tanto de las IPs dentro del campus y de las IPs fuera del campus. La línea verde representa el tráfico TCP dentro del campus, la línea roja representa el tráfico TCP fuera del campus.

4.3.2 Anomalías en paquetes UDP

El front-end coge paquetes UDP en forma de tuplas durante el periodo de recolección de datos, aproximadamente ocurre cada 30 segundos. Al final de cada periodo, Ourmon revisa la tupla y calcula el peso de trabajo usando una fórmula que será detallada más adelante. Además el peso de trabajo del UDP es agregado a la tupla y es un atributo de segundo orden computado. Las tuplas UDP producidas por el periodo de recolección de datos son llamados UDP Port Report. La tupla tiene la siguiente forma:

(IPSRC, WEIGHT, SENT, RECV, ICMPERRORS, L3D, L4D, SIZEINFO, SA/RA, APPFLAGS, PORTSIG).

La clave lógica en esta tupla es la dirección IP IPSRC (IP Source).

- **SENT y RECEIVED** son contadores de los paquetes UDP enviados desde/hasta el host en cuestión.
- **ICMPERRORS** es un contador de tipos distintos de errores ICMP que pueden haber, normalmente la mayoría de los errores se dan

porque el destino es inalcanzable.

- **L3D** es el contador de la capa 3 de direcciones IP destino durante el periodo recolectado.
- **L4D** es el contador de los puertos UDP destino.
- **SIZEINFO** es un histograma de tamaño de paquetes enviados a nivel capa 7.
- **SA/RA** es un promedio de la carga útil del tamaño de los paquetes UDP en capa 7 enviados por el host (SA) y recibidos por el host (RA).
- **APPFLAGS** es un campo programable basado en expresiones regulares y es usado para inspeccionar el contenido en capa 7.

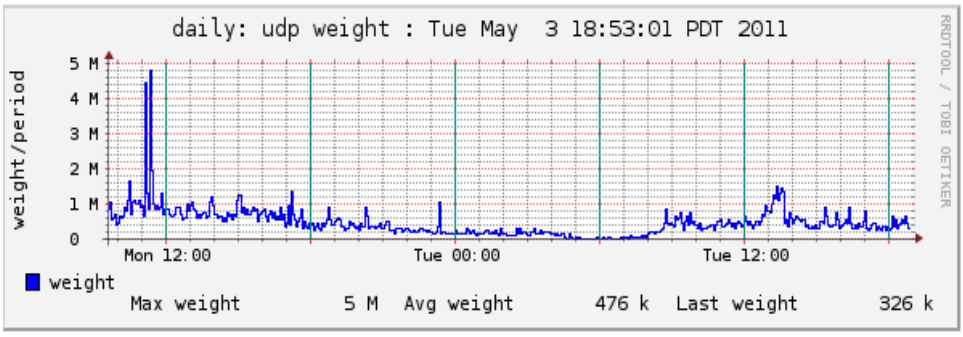


Figura 4.4 Grafica de la actividad UDP en el campus de Portland, OR.

En la figura 4.4 se notifica el peso del tráfico UDP por período. El peso máximo es un peso determinado por el administrador de redes.

Para calcular el Work Weight se utiliza la siguiente fórmula:

*Peso de trabajo = (SENT * ICMPERRORS) + REC*

En un período de recolección de datos, contamos los paquetes UDP enviados por el host y se lo multiplica por el número total de errores ICMP que han sido retornados al host.

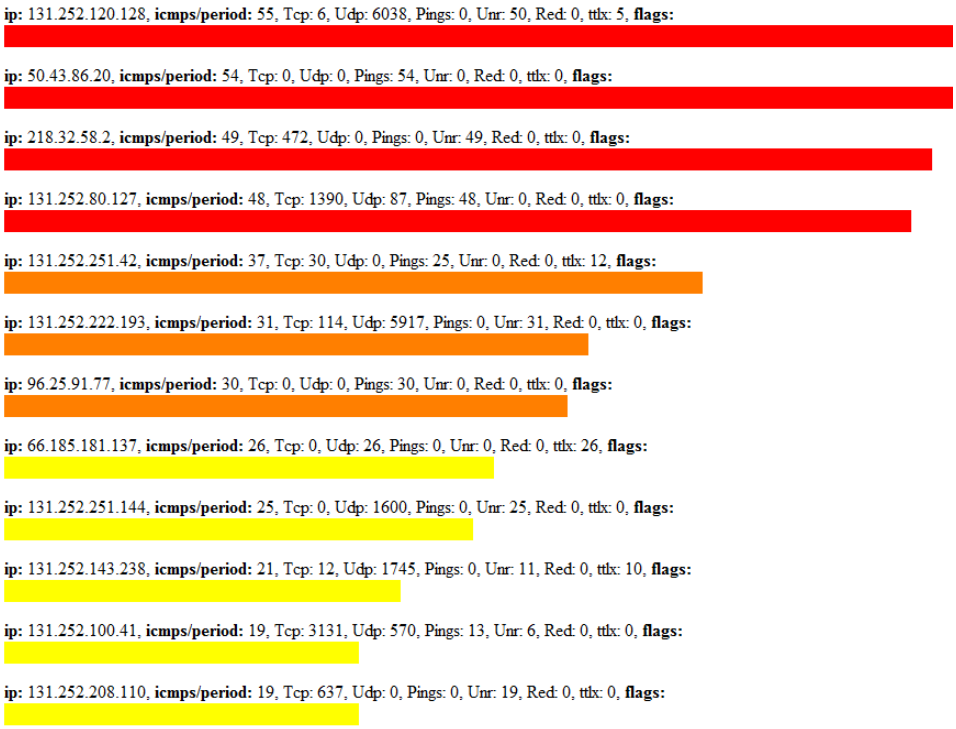


Figura 4.5 Lista con las IPs y sus respectivos ICMP Errors.

Los errores de ICMP, como lo habíamos mencionado son parte de aquellos hosts que no han logrado ser alcanzados. Si existen conteos

bastante altos de errores (ICMPERRORS), el peso de trabajo será muy alto. Según los estudios hechos por la Universidad de Berkeley el peso de trabajo en la red del campus de la universidad de Portland, OR, son alrededor de 75000, si pesos de trabajo extremadamente altos se podría tratar de un ataque DDoS o de un scan **(17)**.

4.4 Integración final de la solución IDS/IPS Empresarial

Para la Solución Empresarial propuesta en esta tesina de seminario, la información almacenada por Ourmon está en un archivo llamado 'tcpworm.txt' con el host o hosts sospechosos.

Con un script creado por nosotros, se realizan lecturas de este archivo cada minuto, se accede a la información relevante del tcpworm.txt que es la dirección IP y en caso de existir un posible ataque en curso, se crea un nuevo set de reglas para ser leído por Suricata en modo IDS que será guardado en un archivo llamado 'Ourmon-anomaly.rules' donde estarán todas las IPs aprendidas por Suricata desde Ourmon.

```
alert ip -SUSPICIOUS_IP- any -> $HOME_NET any (
msg:"Anomaly Detected, possible worm attack";
classtype: anomaly-ourmon; sid:1100000; rev:1;)
```

Figura 4.6 Regla generada para alertar el host sospechoso detectado por ourmon

El administrador recibirá una alerta cuando se notifique de un posible ataque anómalo ya sea un gusano o un scanner, dependerá del administrador y su expertise, tomar medidas ya sea ubicando a esa IP alertada por nuestra solución en un Black List o si conoce que dicha IP es segura, ubicarlo en un White List.

Las reglas generadas por este método tienen acción ALERT, esto debido a que este algoritmo está sujeto a falsos positivos, si bien es cierto son sospechosas no implica que siempre será un ataque en la red. Por lo tanto hemos considerado para una detección eficiente de posibles ataques anómalos, dentro de nuestra Solución. Poner estas reglas en modo DROP podría representar el bloqueo de direcciones IPs que incluso son parte de algún escaneo, bloquearlas perjudicaría de alguna forma el entorno de red.

Una desventaja de esta solución está en que la regla generada no tendrá efecto hasta que se reinicie Suricata. Se puede programar un reinicio automático cada cierto tiempo. Dado que esta solución se la sugiere usar en

modo IDS, no habría mucho problema en el reinicio; e incluso en modo IPS, con el sistema de recuperación de fallos, el reinicio sería transparente para la red, aunque se dejaría la red expuesta a ataques durante unos 20-30 segundos que dura el reinicio del motor.

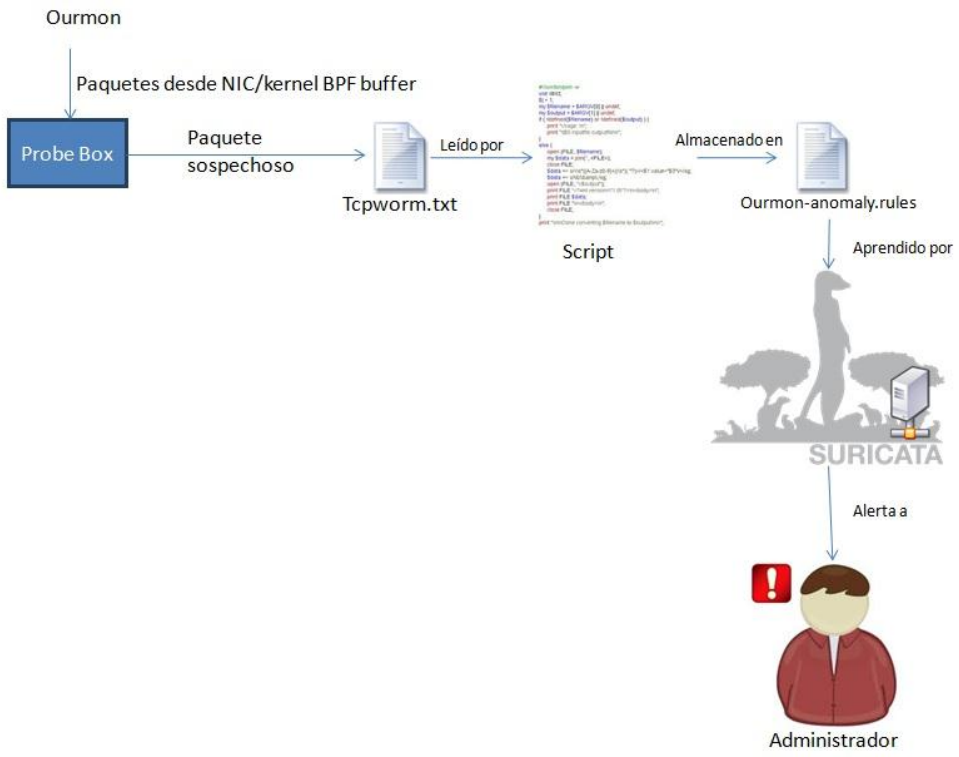


Figura 4.7: Diagrama de módulo de detección de anomalías y auto aprendizaje

CAPITULO 5

DISEÑO Y EJECUCIÓN DE PRUEBAS DE RENDIMIENTO DE SURICATA

Dado que Suricata es un motor de Detección y Prevención de Intrusos (IDS/IPS) hay que tomar en cuenta que estos dos modos diferentes de funcionamiento también requieren de prioridades y objetivos de pruebas diferentes.

Como se explicó en el capítulo 1, el IDS es un dispositivo que físicamente se encuentra en modo espejo (mirror) recibiendo una copia de todos los paquetes que viajan por la red. Su capacidad de procesamiento debe como mínimo

igualar la carga de red promedio. En este modo la latencia no es una prioridad y valores de hasta 1 minuto son aceptables en ciertos casos. Para cumplir estos requisitos en caso de ráfagas de tráfico los IDS poseen una mayor cantidad de memoria para buffers para prevenir estas situaciones. Dado que la latencia no es una prioridad, no hay problema en asignar buffers de mucha memoria.

Por otro lado, los IPS son dispositivos que se encuentran físicamente en modo Inline, observando, procesando y filtrando todo el tráfico de la red. Todo este procesamiento de paquetes, por más rápido que sea , requiere tiempo y agrega obligatoriamente latencia en la red, lo cual termina afectando al throughput. En el caso del IPS, la capacidad de procesamiento debe igualar el tráfico pico de la red y la latencia debe estar a la par con la conexión más rápida en la red. Si el throughput en el IPS es menor al pico de la red, el dispositivo se convierte en un cuello de botella. El uso de buffers de mucha memoria para combatir las ráfagas no es aceptable para los IPS ya que producirá un incremento en el valor de latencia que afectará el tiempo de respuesta de las aplicaciones.

Dado lo expuesto en los párrafos anteriores el desafío de los motores IDS/IPS, en cuanto a pruebas de throughput, está realmente en la parte de Prevención de Intrusos, pues es la que determinará si el sistema es apto o no para soportar la red en la que se lo necesita.

Para las pruebas de desempeño es necesario inundar al IPS con mezcla de tráfico real, debido a que parámetros como: tamaño de paquetes, distribución de protocolos, contenido del paquete, número de sesiones y sesiones por segundos, número de paquetes y paquetes por segundo, duración de sesiones, etcétera, todos tienen un impacto en el desempeño del motor. En un ambiente común, el tráfico lo componen en su mayoría una combinación de paquetes TCP y UDP; por lo que no sería "real" probar con un tipo de tráfico a la vez.

5.1 Diseño de pruebas de rendimiento de la solución empresarial.

5.1.1 Hardware usado para las pruebas.

Suricata:

- Motherboard: Intel Corporation DP55WB
- Procesador: Intel(R) Core(TM) i7 CPU 870@2.93GHz
- Memoria RAM: 8 Gigas
- Tarjetas de Red: 3 Intel 82572EI Gigabit Ethernet

Adicionalmente se usaron 3 computadoras adicionales de menores recursos para la generación de tráfico.

Todas las computadoras tienen tres interfaces de red, dos para comunicación entre ellas y una interfaz para administración. Las dos interfaces de comunicación están conectadas a un Switch Cisco Catalys C3560G. Se crearon dos vlans para forzar que el Suricata esté en el medio de la comunicación entre dos computadoras de prueba (Sin las Vlans, las comunicación únicamente pasaría por el Switch al ser el dispositivo de capa 2 que está antes de Suricata).

5.1.2 Herramientas utilizadas para Pruebas

Tomahawk

Es una herramienta opensource para realizar pruebas de throughput y capacidades de bloqueo para sistemas de prevención de Intrusos.

Tomahawk permite someter a los NIPS a pruebas generando un tráfico conformado por una mezcla de protocolos simulando un tráfico real. Para esto, repite capturas de paquetes previamente obtenidos con herramientas como Wireshark, tcpdump o ngrep.

A diferencia de herramientas similares como tcpreplay, Tomahawk no solo repite la captura de paquetes, divide el tráfico en función de las direcciones ip fuente y destino y lo clasifica en paquetes de cliente o servidor. Esto es útil para comprobar si un paquete generado por una

interfaz y que se esperaba recibir por otra interfaz ha sido bloqueado por algún dispositivo en medio, en este caso el IPS. Tomahawk Permite también repetir varias veces la captura e incluso correrlas en paralelo para aumentar el tráfico generado.

Un proceso de Tomahawk puede generar hasta 450 Mbps de tráfico y entre 25 y 50 mil conexiones por segundo. Con dos computadoras no tan robustas conectadas a través de un switch basta para generar hasta 1 Gbps de throughput y hasta 90 mil conexiones por segundo para poner al NIPS a una completa prueba de estrés.

Además esta herramienta permite repetir muestras de tráfico de algún ataque conocido con la intención de comprobar si el NIPS fue capaz o no de detener dicho ataque **(18)**.

hping3

Es una herramienta multi-usos para generar paquetes TCP/IP. Provee flexibilidad en el manejo y customización de paquetes y es ampliamente usado en pruebas de desempeño de red, servidores y dispositivos de red **(19)**.

Httperf

Herramienta desarrollada para realizar pruebas de desempeño en los servidores HTTP. Es más usado en pruebas de stress para medir las capacidades de los servidores web **(20)**.

Siege

Simula tráfico real hacia servidores web generando peticiones constantes desde varios navegadores web virtuales. Permite cargar una lista de peticiones web para luego repetirlas simulando tráfico de internet **(21)**.

Iperf

Sirve para medir el desempeño máximo de ancho de banda para tráfico TCP y UDP. Ademas es configurable por lo que también puede ser usado para crear ruido en el canal simulando descargas o streamings de multimedia **(22)**.

5.2 Escenario y topología para las pruebas.

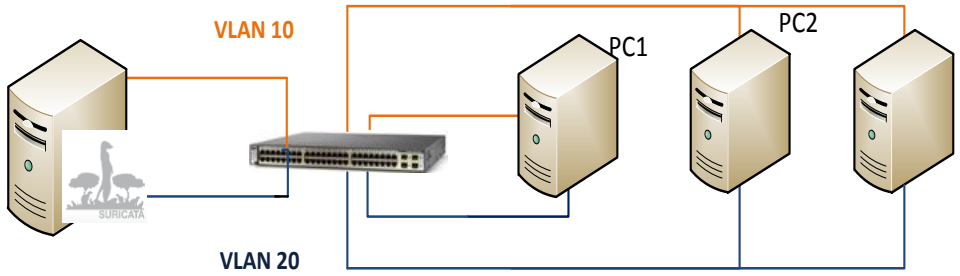


Figura 5.1 Topología para las pruebas

Se instaló perf en las tres computadoras de prueba.

Se instaló Apache2 en una de las computadoras de prueba para responder peticiones solicitadas por httperf y Siege.

Se instaló Tomahawk en 2 computadoras y el resto de herramientas en las 3 por igual.

Se crearon varios escenarios de prueba usando diferentes herramientas de prueba aparte de las mencionadas. Al final para pruebas generales se realizaron varias capturas de diferentes tipos de tráfico para luego ser generadas con Tomahawk. Gracias a la capacidad de Tomahawk de repetir varias capturas en paralelo, fue posible simular desde una captura de pocas computadoras, un tráfico tal como si se tratara de cientos de clientes transfiriendo paquetes.

La Figura 6.1 detalla la topología usada. La separación de la red en VLANS, fue necesaria para obligar a que los paquetes pasen a través del Suricata en lugar de que sean conmutados por el switch.

5.3 Implementación y análisis de resultados de las pruebas.

- Prueba 1: CPU vs Throughput en modo IDS

En modo IDS, Suricata analiza los paquetes capturados con la librería libpcap, dado que los paquetes analizados son solo una copia del tráfico de red no se pudo analizar hasta 1 Gbps de Throughput sin ningún inconveniente.

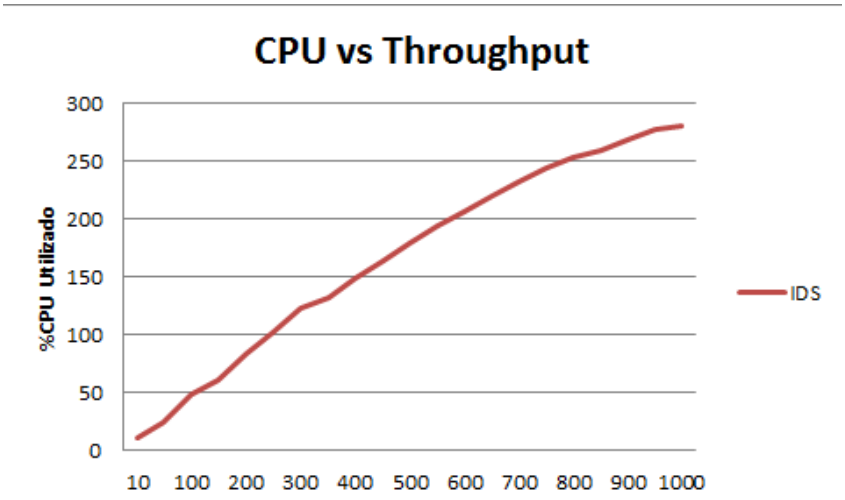


Figura 5.2: Utilización de CPU vs Througput en modo IDS

En modo IPS, Suricata lee los paquetes que fueron encolados en iptables con el módulo de NFQUEUE. Desde la versión 1.0.1 es posible encolar los paquetes en más de una cola para ser leídos por un mismo proceso de Suricata por lo que se probó el desempeño de procesamiento de Suricata usando diferente número de colas.

- Prueba 2: CPU vs Throughput en modo IPS

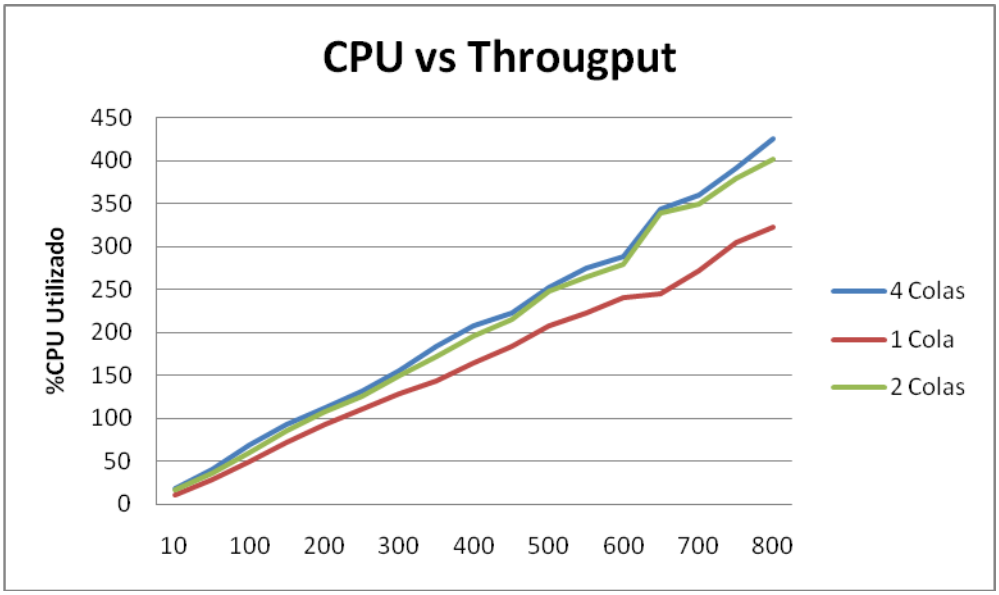


Figura 5.3: Utilización de CPU vs Througput en modo IPS

Se aprecia que al utilizar más colas aumenta el procesamiento del servidor, sin embargo aumenta el Througput máximo del motor. La diferencia entre 2

colas y 4 colas es casi indiferente aunque se aprecia un aumento de throughput máximo en este último. A partir de 4 colas el procesamiento aumenta un poco pero no aumenta el throuhput máximo por lo que no tiene sentido aumentarle más.

Esta limitante no tuvo que ver con saturación en los procesadores. En las figuras 6.2 y 6.3 se muestra la salida de ejecutar los comandos pidstat y top respectivamente. Ninguno muestra saturación en los procesadores del servidor.

TGID	TID	%usr	%system	%guest	%CPU	CPU	Command
21634	-	169.00	146.00	0.00	315.00	1	suricata
-	21634	0.00	0.00	0.00	0.00	1	|__suricata
-	21641	3.00	7.00	0.00	10.00	5	|__RecvNFQ-Q1
-	21642	2.00	6.00	0.00	8.00	6	|__RecvNFQ-Q2
-	21643	7.00	1.00	0.00	8.00	7	|__RecvNFQ-Q3
-	21644	3.00	7.00	0.00	10.00	4	|__RecvNFQ-Q4
-	21645	15.00	13.00	0.00	28.00	1	|__Decode1
-	21646	7.00	2.00	0.00	9.00	0	|__Detect1
-	21647	5.00	4.00	0.00	9.00	2	|__Detect2
-	21648	8.00	2.00	0.00	10.00	1	|__Detect3
-	21649	10.00	0.00	0.00	10.00	3	|__Detect4
-	21650	8.00	2.00	0.00	10.00	0	|__Detect5
-	21651	11.00	0.00	0.00	11.00	1	|__Detect6
-	21652	7.00	2.00	0.00	9.00	2	|__Detect7
-	21653	5.00	4.00	0.00	9.00	3	|__Detect8
-	21654	7.00	3.00	0.00	10.00	0	|__Detect9
-	21655	7.00	3.00	0.00	10.00	1	|__Detect10
-	21656	9.00	1.00	0.00	10.00	2	|__Detect11
-	21657	10.00	0.00	0.00	10.00	3	|__Detect12
-	21658	10.00	0.00	0.00	10.00	0	|__Detect13
-	21659	10.00	1.00	0.00	11.00	1	|__Detect14
-	21660	6.00	4.00	0.00	10.00	2	|__Detect15
-	21661	5.00	4.00	0.00	9.00	3	|__Detect16
-	21662	2.00	15.00	0.00	17.00	4	|__VerdictNFQ0
-	21663	3.00	19.00	0.00	22.00	5	|__VerdictNFQ1
-	21664	4.00	18.00	0.00	22.00	6	|__VerdictNFQ2
-	21665	2.00	20.00	0.00	22.00	7	|__VerdictNFQ3
-	21666	4.00	4.00	0.00	8.00	3	|__Outputs
-	21667	4.00	2.00	0.00	6.00	4	|__FlowManagerThre
-	21668	0.00	0.00	0.00	0.00	2	|__SCPerfWakeupThr
-	21669	0.00	0.00	0.00	0.00	4	|__SCPerfMgmtThrea

Figura 5.4 Procesamiento de Suricata por Hilos con 4 colas

```
Tasks: 148 total,   1 running, 147 sleeping,   0 stopped,   0 zombie
Cpu0  :  9.8%us, 23.4%sy, 36.6%ni, 30.2%id,  0.0%wa,  0.0%hi,  0.0%si,
Cpu1  : 43.9%us, 18.0%sy,  0.0%ni, 38.0%id,  0.0%wa,  0.0%hi,  0.0%si,
Cpu2  : 36.1%us,  6.5%sy,  0.0%ni, 57.4%id,  0.0%wa,  0.0%hi,  0.0%si,
Cpu3  : 35.5%us,  4.7%sy,  0.0%ni, 59.9%id,  0.0%wa,  0.0%hi,  0.0%si,
Cpu4  :  4.3%us, 13.7%sy,  0.0%ni, 61.2%id,  0.0%wa,  0.0%hi, 20.7%si,
Cpu5  :  6.4%us, 10.6%sy,  0.0%ni, 61.4%id,  0.0%wa,  0.0%hi, 21.5%si,
Cpu6  :  9.7%us, 19.1%sy,  0.0%ni, 67.7%id,  0.0%wa,  0.0%hi,  3.5%si,
Cpu7  : 11.0%us, 20.7%sy,  0.0%ni, 64.9%id,  0.0%wa,  0.0%hi,  3.4%si,
Mem:   8174536k total,  2340340k used,  5834196k free,   133192k buffers
Swap:  1052248k total,        0k used,  1052248k free,   770272k cached

21634 root      20   0 1607m 1.1g 1732 S  338 13.8  37:26.52 suricata
   33 root      20   0     0    0    0 S    0  0.0   6:43.84 events/6
    1 root      20   0  3852  668  568 S    0  0.0   0:03.30 init
  867 root      20   0     0    0    0 S    0  0.0   0:04.82 kjournald
```

Figura 5.5 Procesamiento de Suricata por procesador con 4 colas

5.4 Pruebas del módulo de detección de anomalías.

Hemos puesto al módulo de detección de anomalías en tráfico real para evaluar su efectividad en capturar posibles ataques de gusano y además probar el anomaly detection. Como se ha evaluado en tráfico real, para mantener confidencialidad hemos borrado los últimos 2 octetos de las direcciones IP.

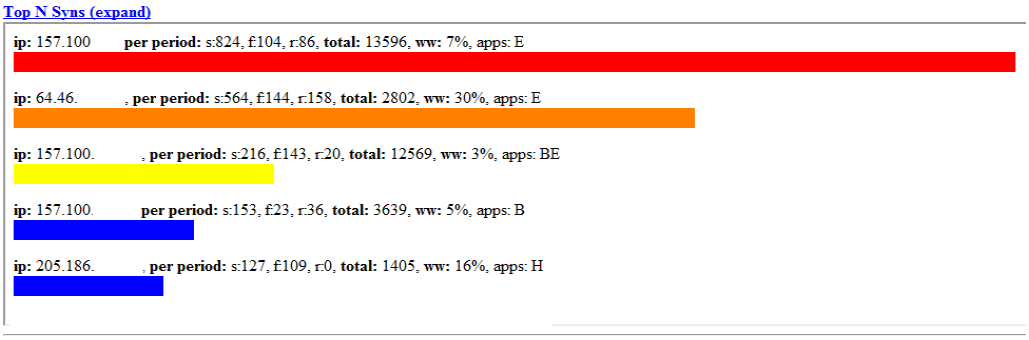


Figura 5.6: Top Syns que entran a la red.

Se observa en la figura 5.6 las direcciones IP con mayor número de Syns (contadores de paquetes) que estuvieron en la red, no necesariamente a mayor cantidad de Syns representaría un posible ataque de gusano o escaneo de red.

TCP worm graph:

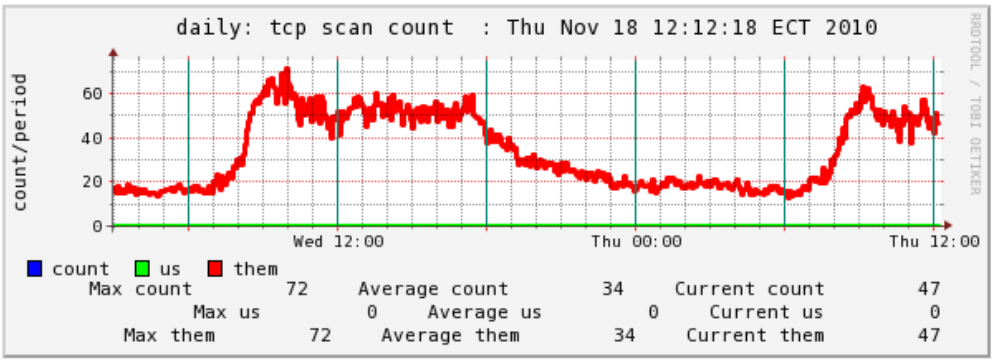


Figura 5.7 Grafo de detección de gusanos utilizando Ourmon.

En la figura 5.7, Ourmon está mostrando un grafo de gusano en base al peso de trabajo que está ocurriendo en un lapso de 24 horas que hemos puesto a Ourmon en prueba, a mayor peso de trabajo, más es la probabilidad de que un escaneo o ataque de gusano se esté dando.

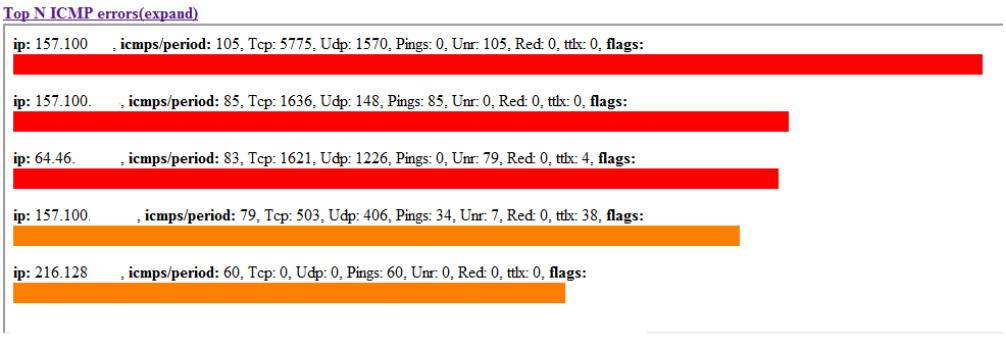


Figura 5.8 Lista con las IPs y sus respectivos ICMP Errors.

En la figura 5.8 tomamos datos de la cantidad de ICMP Errors para detectar ataques usando el protocolo UDP, afortunadamente no existieron ataques UDP, lo corrobora el gráfico 5.9.

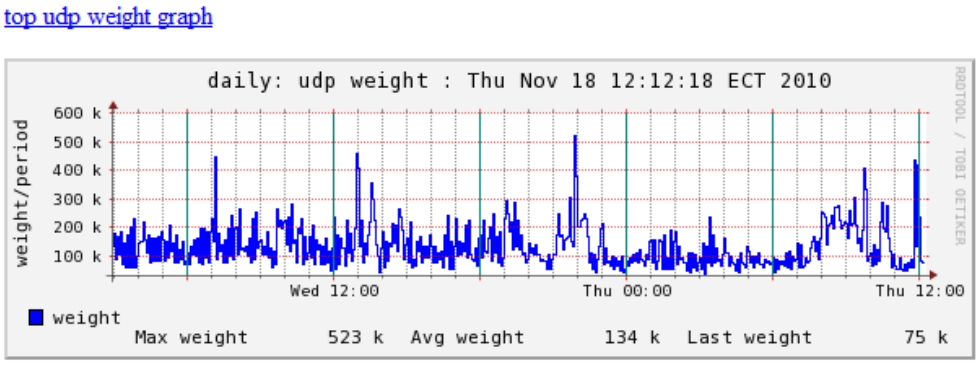


Figura 5.9: Gráfico de peso de trabajo en el protocolo UDP.

Por último en el archivo tcpworm.txt se sacaron datos de las direcciones IP que tenían peso de trabajo de 100%, a estas direcciones se las mandaron al archivo de rulesets “Ourmon-anomaly.rules”, con esto nuestra solución

empresarial aprendió que estos paquetes enviados a la red contenían efectivamente tráfico anómalo.

```
start log time          : instances: DNS/ip : syns/fins/resets/total pkts(%work info) total counts
end log time        :  portsig info (max=5) sampled port signatures format: [dst port, pkt count] [port, count] ...
--------------------------------------------------------------------------------
Thu Nov 18 00:00:04 2010: 1368: 75.126.     :337404:0:22603:359967(100%:W)
Thu Nov 18 11:22:02 2010:       portsigs(5/5): [1086,2,1080,3,1041,2,1157,2,1138,1,1269,1,1205,2,1192,3,1151,2,1106,4,]
[1168,2,1255,2,1184,4,1092,1,1229,1,1139,5,1259,3,1260,2,1237,1,1232,1,] [1173,3,1231,2,1198,4,1089,2,1273,3,1264,1,1048,3,1268,2,1176,1,1105,2,
[1060,2,1097,1,1093,3,1154,1,1227,3,1039,3,1270,4,1184,1,1139,2,1144,1,] [1060,2,1097,1,1093,3,1154,1,1227,3,1039,3,1270,4,1184,1,1139,2,1144,1,
Thu Nov 18 00:00:04 2010: 865: 79.142.      :138772:0:4152:142810(100%:W)
Thu Nov 18 07:11:17 2010:       portsigs(5/5): [10847,2,10064,2,5517,2,3140,2,7101,2,2905,2,9018,2,7473,2,5403,2,6586,1,]
[1397,2,9462,1,3724,1,5572,1,11000,1,5190,1,10546,1,11019,1,7814,1,10081,1,] [7771,1,2758,1,10871,1,10746,2,9100,1,1162,1,6761,1,5355,1,10916,1,
[8534,2,3615,2,10114,2,9802,2,5607,1,5087,1,8963,1,8452,1,1884,1,8333,1,] [8534,2,3615,2,10114,2,9802,2,5607,1,5087,1,8963,1,8452,1,1884,1,8333,
Thu Nov 18 00:00:04 2010: 264: 97.74.       :124270:0:6400:130550(100%:W)
Thu Nov 18 02:10:04 2010:       portsigs(5/5): [32256,10,26624,9,6656,11,19969,9,40960,13,35328,11,48640,10,7168,12,24065,8,31232,9,]
[11777,11,49665,11,15432,9,6656,9,14849,12,26624,13,15873,7,48640,7,21505,13,35328,6,]
```

Figura 5.10 Archivo tcpworm.txt con las IPs sospechosas

En cuanto al rendimiento de procesador y memoria, Ourmon no representa mucho consumo. La figura 5.11 muestra una captura del comando ‘top’ mostrando ourmon en ejecución y su consumo de recursos.

```
top - 23:23:18 up 15 days,  6:36,  3 users,  load average: 0.21, 0.14, 0.10
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.6%us,  0.1%sy,  0.0%ni, 98.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8301656k total,  2766700k used,  5534956k free,   310040k buffers
Swap:  5406716k total,        0k used,  5406716k free,  1835844k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2657 root      20   0  2020  448  372 S  0.3  0.0   9:38.22 hald-addon-stor
 2819 mysql     20   0 2082m 356m 5560 S  0.3  4.4  51:42.87 mysqld
11449 root      20   0 13920  11m  880 S  0.3  0.1   0:03.09 ourmon
15553 root      20   0  2380  964  704 S  0.3  0.0   0:02.71 top
    1 root      20   0  2116  596  504 S  0.0  0.0   0:07.71 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
```

Figura 5.11 Rendimiento de procesador y memoria de Ourmon

5.5 Conclusiones de los resultados de las pruebas.

Ya habíamos mencionado que en modo IPS Suricata va a tener un througput limitado, no por la capacidad de la interfaz y medio de transmisión sino por qué tan rápido el motor pueda procesar los paquetes. Este valor

máximo de througput va a variar de red en red debido al flujo de paquetes y también al tipo de tráfico. Por ejemplo, una red con gran cantidad de tráfico TCP va a necesitar que Suricata consuma más memoria manteniendo el estado de las sesiones.

Paras redes FastEthernet Suricata tiene un buen performance como IPS, pero en redes GigabitEthernet ya empieza a limitar el throughput a pesar de su arquitectura multithreading y su Hardware, ya que como se ve en las gráficas no se ha saturado. En los Backbones de redes más grandes se debe contar con un dispositivo comercial más robusto para proteger a ese nivel con reglas más generales y tener servidores con Suricata protegiendo segmentos de red medianos con reglas más específicas.

Dado que se contaba con 8 procesadores, el máximo valor de Porcentaje de Utilización de CPU que se puede alcanzar es de 800%. Eso significa que a pesar de que el procesador no está saturado, el throughput sigue limitado. Esto es debido a la latencia producida por el IPS al procesar los paquetes, por lo que ni aumentando CPU, memoria, o tamaño de buffers se puede mejorar este valor. Para sobrepasar este valor se necesita de mejoras en el código de Suricata.

Correr Suricata en modo Inline a altas velocidades en una máquina Linux sin optimizar va a disminuir de una manera notable el desempeño del motor y lo más probable es que al poco tiempo de correr el sistema saldrán los registros se llenaran con un mensaje de error indicando que se han descartados paquetes debido a que se han llenado las colas o los buffers.

Para evitar esto hay que modificar la primera opción del archivo de configuración:

max-pending-packets: 5000

#max-pending-packets: 50

Por defecto nos encontramos con un valor de 50, muy pequeño, por lo que hay que subirlo a un valor más apropiado para mayor tráfico. Por supuesto que este incremento en el desempeño no es gratis, se requiere de mayor memoria RAM por lo que no es bueno considerarlo un valor fijo, sino más bien adaptarlo a las capacidades de Hardware con las que se cuenta.

También será necesario aumentar el valor del buffer que usa NFQ. Estos valores ya son de configuración de Kernel de Linux. Por defecto viene un valor relativamente pequeño que a grandes velocidades hacían que Suricata también bote errores por haber rebosado los buffers de memoria. Para evitar esto se les ha aumentado también el valor:

sysctl -w net.core.rmem_default='8388608'

sysctl -w net.core.wmem_default='8388608'

Los siguientes valores aumentan los buffers destinados a tráfico tcp:

sysctl-w net.ipv4.tcp_wmem='1048576 4194304 16777216'

sysctl -w net.ipv4.tcp_rmem='1048576 4194304 16777216'

Logramos también con el módulo de detección de anomalías integrado a nuestra solución empresarial detectar ciertas direcciones IP que estaban produciendo tráfico anómalo dentro de una red a tiempo real, esto sirvió para el motor Suricata aprender de ellas y generar alertas y acciones idóneas contra este tipo de ataque que sin ayuda de la detección de anomalías no hubiese sido posible encontrar a tiempo.

Lastimosamente Suricata debe reiniciarse cada cierto tiempo para tener conocimiento de las nuevas reglas añadidas por el detector de anomalías así que no se puede hablar de un autoaprendizaje en tiempo real.

CAPITULO 6

PROPUESTAS DE MEJORAS PARA LA SOLUCIÓN EMPRESARIAL ACTUAL

6.1 Interfaz gráfica de gestión y control remoto

La idea es ser capaz de controlar, administrar y monitorizar no sólo uno sino varios sensores remotos. Para esto se necesita una aplicación que posea

una arquitectura centralizada, es decir que se pueda administrar varios motores de suricata desde la misma aplicación.

Una recomendación para el sistema propuesto es montar un Servicio Web o una aplicación de cliente en el lado de los Suricata que se comunique con una aplicación remota (podría ser un aplicación web).

Debido a que las firmas de seguridad las estamos almacenando en una base de datos y las alertas de suricata por medio de barnyard también se pueden guardar en una, solo nos debemos preocupar por mantener una comunicación de presencia entre el servidor central y los secundarios y enviar señales de inicio o detención de servicios.

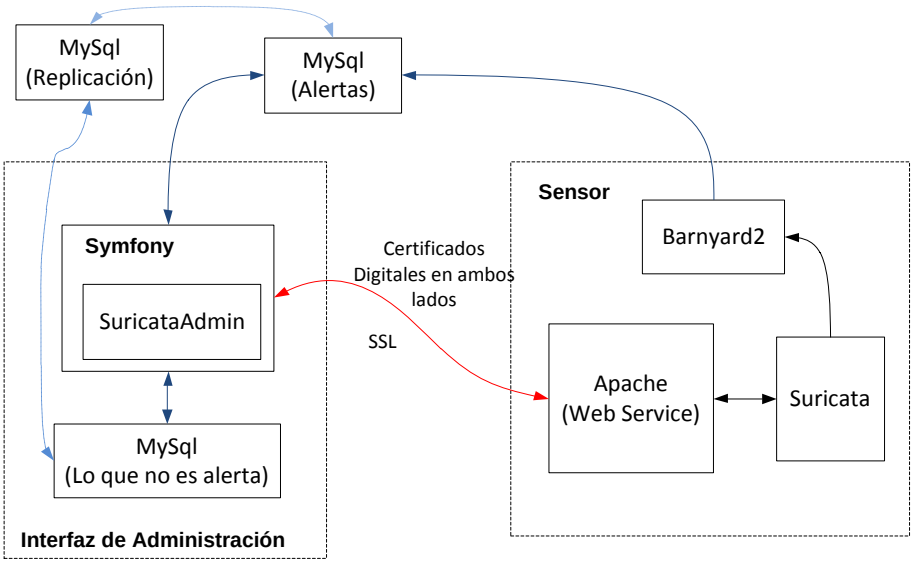


Figura 6.1: Diagrama de bloques para una interfaz centralizada

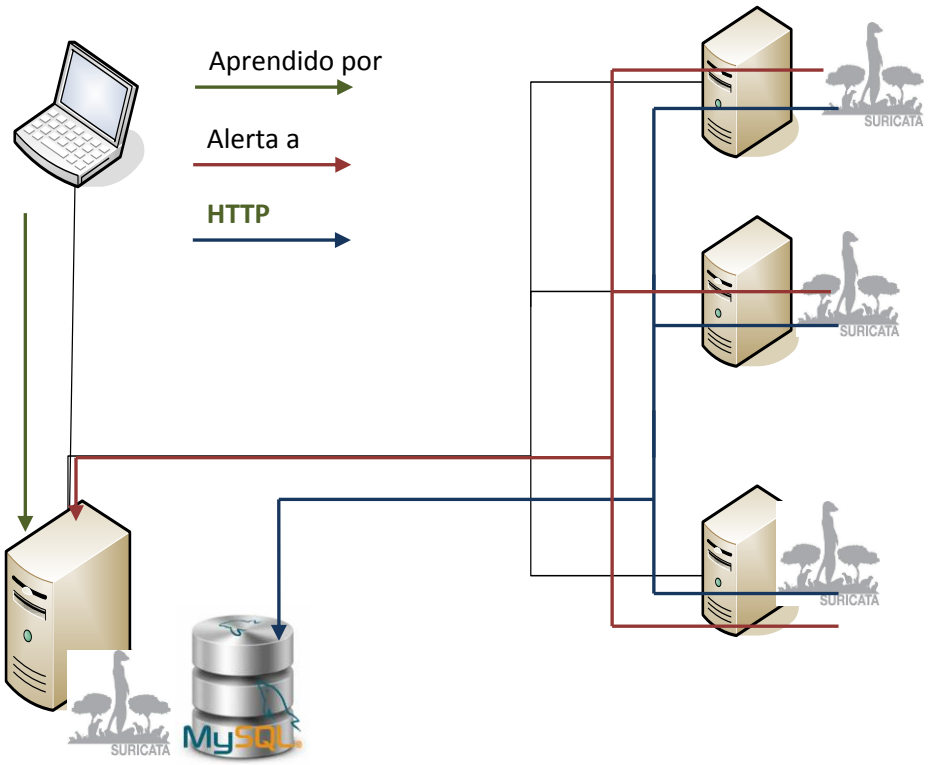


Figura 6.2: Diagrama Web Service para una interfaz centralizada

La comunicación con la interfaz será a través de http. El frontend de administración no necesariamente tiene que estar en el servidor central. Puede estar en una máquina ajena a todo suricata en la red. Lo mismo aplica a la base de datos

Desde la interfaz de administración remota el usuario es capaz de ver todos los servidores con suricata activos en la red, identificando tanto al principal como a los secundarios. Es posible en cada uno ver las características del servidor, iniciar y detener el servicio de suricata, observar las alertas y administrar las firmas de seguridad.

6.2 Módulo de monitoreo para Centro de Operaciones de Redes.

Este módulo debería mostrar gráficas en tiempo real con reportes periódicos configurables. Los reportes deben ser almacenados para futuras referencias y a su vez enviadas por correo electrónico, o cualquier otro mecanismo, al administrador.

Debería ser capaz también de configurar alertas prioritarias que alerten inmediatamente al administrador cuando sean activadas.

6.3 Módulo de Detección Reactiva para Suricata en modo IDS

Suricata en modo IDS actualmente funcionan de una manera pasiva, es decir que únicamente alerta sobre el evento sospechoso y el administrador de red es quién debe tomar acciones oportunas.

Algunos IDS son capaces de reaccionar a los eventos mediante reseteo de conexiones TCP (en caso de escaneos de red o ataques de inundación), e

incluso enviar señales a los Firewalls para bloquear o redirigir paquetes en base a su dirección IP origen.

Si bien es cierto, Suricata en modo IPS ya funcionan de una manera reactiva ante ataques de red, bloqueando los paquetes y terminando sesiones; como se vio en la sección 1.1.3 sobre *Diferencias entre IDS e IPS*, los IDS pueden tomarse algo más de tiempo para alertar y tomar acciones aumentando el througput máximo permitido por el dispositivo.

6.4 Otras propuestas de mejoras

Suricata por ser relativamente nuevo tiene aún mucho camino por recorrer, especialmente en lo que a código se refiere, pues la mayoría de las aplicaciones desarrolladas, por no decir todas, son aplicaciones que en primer lugar fueron desarrolladas para trabajar en conjunto con snort, y que gracias a la compatibilidad de Suricata han podido adaptarse a este motor IDS/IPS.

Como propuestas adicionales de mejoras pueden ser:

- Integración del módulo de Detección de Anomalías al motor.
- Integración del módulo de Recuperación de Fallos y alta Disponibilidad al motor.

- Análisis y optimización del código para aumentar el througput máximo en modo IPS limitado por la latencia que produce el motor.

CONCLUSIONES

1. Los IDS/IPS son herramientas necesarias para tener una protección completa de cualquier arquitectura computacional en los negocios. Por lo valioso que es la información y por la infinidad de ataques que existen aprovechándose de las vulnerabilidades de muchos sistemas, se acentúa la necesidad de protegerla tomando las medidas ya mencionadas.

2. Se estableció patrones de medición de rendimiento utilizando libcaps de tráfico pesado para evaluar la capacidad de Suricata de analizar dicho tráfico sacando provecho la tecnología multi-hilos que lo caracteriza.

3. Todas las herramientas utilizadas en esta tesina de seminario de grado han sido de código abierto, demostrando la compatibilidad entre ellas y consistencia

durante el desarrollo de esta tesina de seminario. No se presentaron problemas de conexión y nos permitió conocer más a fondo sobre las bondades del código abierto.

4. Se conoció algoritmos de detección de anomalías propuesto por Ourmon para contrarrestar scans intensivos y posibilidades de infección de gusanos, que pueden ser utilizadas por Suricata como oportunidad de mejora para una protección más completa del negocio.

RECOMENDACIONES

1. Para poder trabajar con el sistema, es necesario definir políticas y procedimientos de seguridad dentro de la empresa. Esto es una iniciativa para la protección de la información que es ayudada con herramientas como la que hemos tratado en esta tesina de seminario de grado.

2. Se recomienda capacitar al personal encargado del mantenimiento de la solución IDS/IPS de la empresa, pues se requiere conocimientos más allá de saber manejar un Firewall.

3. Se recomienda mantener actualizado los rulesets debido a que cada día se puede registrar un nuevo ataque de diferente moderación. Cada vez que exista un ataque en la red, EmergingThreats se encargará de neutralizarlo con la regla

correspondiente.

4. Se debe concientizar más a los empresarios sobre la existencia y las bondades de la Seguridad Informática y de la amenaza constante que existe en las redes hacia la información, que es el mayor activo de la empresa.

5. Se debe realizar nuevas pruebas de rendimiento conforme los avances tecnológicos sigan llegando al mercado, debido a que Suricata, siendo un IDS/IPS que basa su óptimo trabajo en hilos, puede tomar ventaja en estos y seguir siendo la mejor opción en cuanto a los IDS/IPS de código abierto.

6. Esta solución es recomendable para pequeñas y medianas empresas que no disponen de capital para invertir en un IPS comercial, las capacidades de Suricata, siendo de código libre, cubre muy bien las necesidades de protección de red para estas empresas.

Glosario

Throughput: Es el volumen de información que fluye en las redes de datos. Significativo en el almacenamiento de información y sistemas de recuperación de la información, en los cuales el rendimiento en unidades como accesos por hora.

Latencia: En redes conmutadas es una medida del tiempo que le toma a un paquete desde que su origen hasta su destino. Cada dispositivo de red que el paquete tiene que atravesar, aumenta el valor total de latencia, el cual viene a ser la suma de todas las latencias en el recorrido del paquete.

Rulesets: Colección de reglas que definen una o varias opciones que son comparadas con los paquetes procesados para determinar si se trata de un intento de ataque conocido. También se define que acción se debe tomar: Alertar, descartar, dejar pasar, registrar en log o redirigir a otra regla. Por lo general, las rulesets se obtienen de organizaciones mundiales que se encargan

de actualizarlas periódicamente según van apareciendo nuevos tipos de ataques.

Firmas de Seguridad (Signatures): Es una huella única que identifica un determinado ataque de red. Se suele mezclar este término con el de las reglas que conforman las rulesets debido a que una regla es definida mediante una firma de seguridad. Por ejemplo, para identificar un escaneo de red por nmap se busca por un contenido dentro del paquete que siempre lleva cuando se usa dicho software, el contenido a buscar dentro del paquete es lo que se conoce como firma de seguridad.

Data Leakage Protection: Sistemas que identifican, monitorean y protegen datos que están siendo usados. Se lo hace mediante inspección, análisis contextual de seguridad de las transacciones y con un framework de manejo centralizado. Dichos sistemas están diseñados para detectar y prevenir uso no autorizado de información confidencial.

Thread: Es un subproceso que permite que una aplicación realice varias tareas concurrentemente. Un conjunto de hilos compartiendo los mismos recursos se conoce como proceso. Cuando se trabaja dentro de arquitecturas multi-núcleos, es posible asignar diferentes hilos del mismo proceso a diferentes núcleos del procesador, aprovechando de mejor manera la capacidad total del sistema.

Trama: Es una unidad de envío de datos. Normalmente una trama consta de una cabecera, datos y cola. En la cola suele estar algún chequeo de errores, en la cabecera habrá campos de control de protocolo. La parte de datos es la que quiere transmitir en nivel de comunicación superior, normalmente en capa de red. En el modelo OSI, la trama es la unidad de datos de protocolo de la capa de 2 o de Enlace de datos.

Stream: El término es usado en comunicaciones como abstracción para definir un flujo de datos transmitidos en un intervalo de tiempo.

Proxy: Es un programa o dispositivo que realiza una acción en representación de otro. Su finalidad es de permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, o sea, una única dirección IP.

NFQueue: Es un manejador flexible de paquetes que permite tomar decisiones de filtrado desde el espacio de usuario. En teoría, filtrar paquetes desde el espacio de usuario y no desde el kernel como lo hace iptables, es una mala idea pues requiere de un procesamiento extra, pero las ventajas que posee esta técnica debido a la flexibilidad del manejador, por el contrario ser una manera más eficiente y óptima de realizar el filtrado. Se usa particularmente en

escenarios en los que se requiere dinamismo y en el caso de usar iptables requeriría de miles de reglas para implementarlo.

IPFRing: Es un nuevo tipo de socket de red que incrementa significativamente la velocidad de captura de paquetes. Se puede usar en conjunto con tarjetas de red fabricadas especialmente para trabajar con PFring, lo cual mejora mucho más la velocidad de captura. Por el momento sólo es posible utilizarse en suricata en modo IDS.

Libpcap: Es un sistema independiente con una interfaz a nivel de usuario para captura de paquetes. Provee un framework para monitorear la red. La aplicación incluye colección de estadísticas, monitoreo de seguridad, debug de la red, etc.

Packet Sniffer: Es un software destinado para detectar tramas en la red. No son descartadas las tramas no destinadas a la MAC de la tarjeta; de esta manera se puede capturar (sniff) todo el tráfico que viaja por la red. Los packet sniffers tienen varios usos, van desde monitoreo de redes para detectar y analizar fallos hasta robar contraseñas, interceptar mensajes de correo, etc. El software más conocido para realizar packet sniffing es Whireshark.

YAML: Es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, etc. Fue creado con la intención de representar datos como combinaciones de listas, hashes o mapeos y datos

escalares. Cuando se trabaja con un archivo en formato .yaml hay que tener mucho cuidado con los cambios de línea y los espaciados, pues es lo que determina el nivel en el que se encuentra un dato (por ejemplo, dentro de la misma lista).

Web Service: Conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programaciones diferentes y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet, esto llega a conseguirse mediante la adopción de estándares abiertos tales como W3C.

Barnyard2: Es un interpretador de código abierto originalmente diseñado para el formato binario unified2 de Snort. La idea es liberar al IPS/IDS de la carga del manejo de la información de salida. Barnyard se encarga de interpretar los archivos binarios unified2 y mandarlos a una aplicación externa como una front-end o una base de datos, Todo esto en un proceso separado que no interfiere con el análisis de tráfico de Snort o Suricata.

Modo Inline: Configuración física de red en la cual un dispositivo se encuentra ubicado en un lugar tal que toda el tráfico de la red pasa a través de él. Para que Suricata funcione como IPS, es necesario que esté trabajando en Modo Inline,

pues es la única manera implementada para que Suricata pueda descartar paquetes que representan un peligro para la red. El modo Inline demanda más capacidad de procesamiento, memoria y velocidad de captura para garantizar el óptimo funcionamiento de la red y no convertirse en un cuello de botella.

Modo Mirror: Configuración física de red en la cual un dispositivo se encuentra ubicado en un lugar tal que recibe una copia de todo el tráfico de red que por lo general se lo logra mediante puertos especiales de los switches. El modo mirror es la configuración más común para los IDS, pues al ser sólo dispositivos de alertas les basta con recibir una copia del tráfico para analizarlo. Es posible que un IPS funcione en Modo Mirror aunque para ello tiene que trabajar en conjunto con un dispositivo externo que interprete alertas por parte del IPS y tome las medidas requeridas de bloqueo.

Botnets: Botnet es un término que hace referencia a un conjunto de robots informáticos o bots, que se ejecutan de manera autónoma y automática. El creador de la botnet puede controlar todos los ordenadores/servidores infectados de forma remota y normalmente lo hace a través del IRC. Las nuevas versiones de estas botnets se están enfocando hacia entornos de control mediante HTTP, con lo que el control de estas máquinas será mucho más simple

Worms: Es un malware que tiene la propiedad de duplicarse a sí mismo. Los gusanos utilizan las partes automáticas de un sistema operativo que generalmente son invisibles al usuario. Un gusano no procede a alterar archivos de programa (como un virus) sino a residir en memoria, y se duplica a sí mismo. Los gusanos causan problemas en la red aunque sea simplemente consumiendo ancho de banda. Se basan en una red de computadoras para enviar copias de sí mismos a otras terminales dentro de la misma red y son capaces de llevar esto a cabo sin la intervención de un usuario, mediante métodos como SMTP, IRC, P2P, etc.

ANEXOS

ANEXO A

Archivo spec de configuración

```
%define _topdir       /home/suricata
%define name          suricata
%define release       1
%define version       1.1
%define buildroot     %{_topdir}/%{name}-%{version}-root

BuildRoot:            %{buildroot}
Summary:              GNU suricata
License:              GPL
Name:                 %{name}
Version:              %{version}
Release:              %{release}
Source:               %{name}-%{version}.tar.gz
Prefix:               /usr
Group:                Development/IPS

%description
Instalador de IDS/IPS Suricata hecho por OISF (Open Information Security
Foundation).

%prep
%setup -q

%build
./configure --enable-nfq
make

%install
make install prefix=$RPM_BUILD_ROOT/usr

%files
%defattr(-,root,root)
/usr/bin/suricata
```

/usr/include/htp/bstr.h
/usr/include/htp/dslib.h
/usr/include/htp/hooks.h
/usr/include/htp/htp.h
/usr/include/htp/htp_decompressors.h
/usr/include/htp/utf8_decoderh
/usr/lib/libhtp-0.2.so.1
/usr/lib/libhtp-0.2.so.1.0.2
/usr/lib/libhtp.a
/usr/lib/libhtp.la
/usr/lib/libhtp.so
/usr/lib/pkgconfig/htp.pc

ANEXO B

Estructura de Base de Datos

Entidades y atributos de la base de datos

Entidad	**Atributos**
Entidades descriptivas necesarias para la administración del sitio	
Parámetros_generales	id, tipo_id, descripción, estado, orden.
Perfil	Id_perfil, nombre, descripción, estado, id_usuario, url
Persona	Id_persona, nombres, apellidos, ciudad, cedula, teléfono, fecha_nacimiento, fecha_registro
Reference	Ref_id, ref_system_id, ref_tag
Reference_system	Ref_system_id, ref_system_name
Rulesets	Id, acción, protocolo, ip_origen, puerto_origen, ip_destino, puerto_destino, opciones, perfil, estado, dirección
Usuario	Id_usuario, login, clave, alias
Base_roles	Role_id, role_name, role_desc
Base_users	Usr_id, usr_login, usr_pwd, usr_name, role_id,

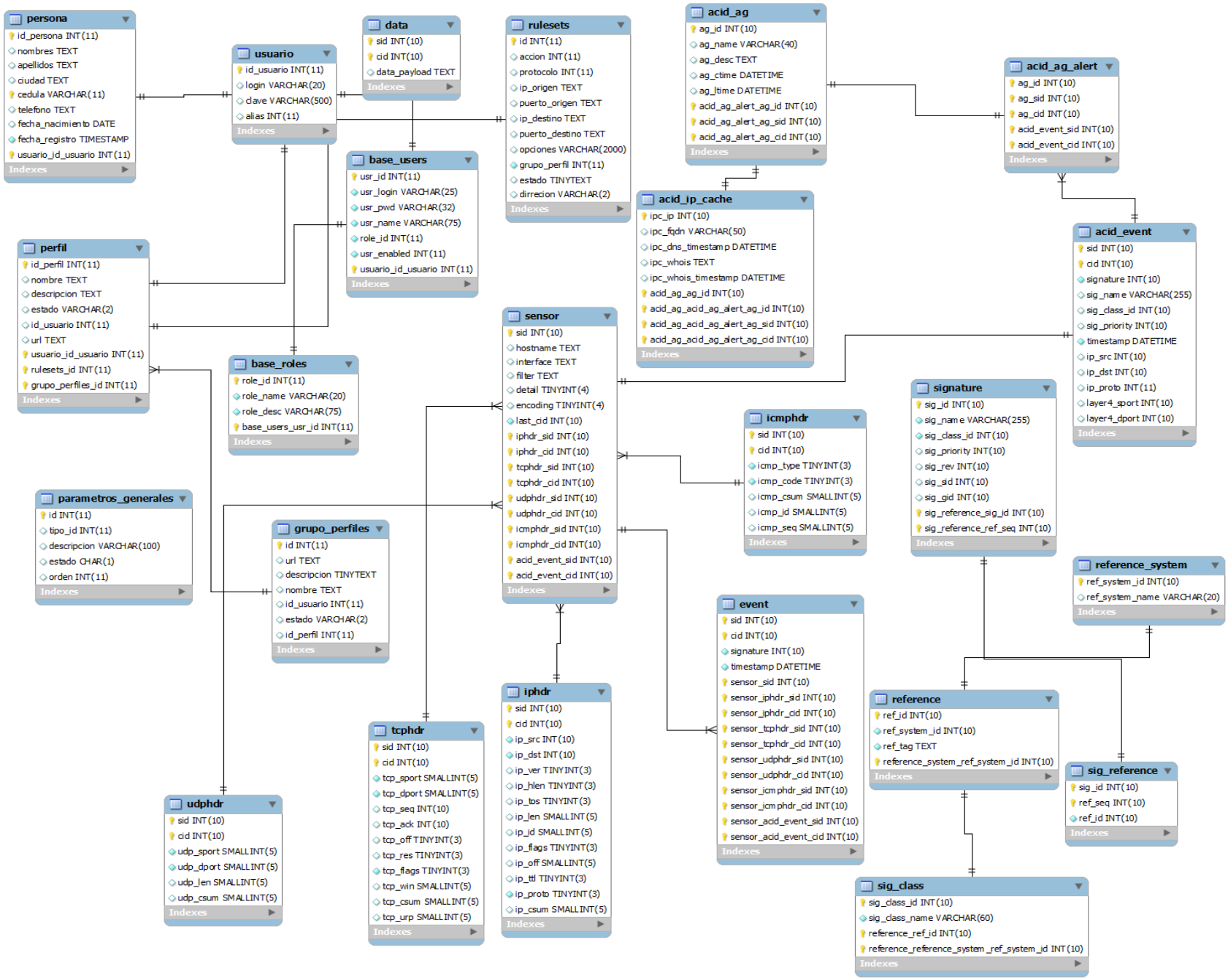
	usr_enabled.
Grupo_perfiles	Id, url, descripcion, nombre, id_usuario, estado, id_perfil
Entidades descriptivas necesarias para el manejo del motor Suricata	
Sensor	Sid, hostname, interface, filter, detail, encoding, last_cid
Sig_class	Sig_class_id, sig_class_name
Sig_reference	Sig_id, ref_seq, ref_id
Signature	Sig_id, sig_name, sig_class_id, sig_priority, sig_rev, sig_sid, sig_gid
Tcp_hdr	Sid, cid, tcp_sport, tcp_dport, tcp_seq, tcp_ack, tcp_off, tcp_res, tcp_flags, tcp_win, tcp_csum, tcp_urp
Udp_hdr	Sid, cid, udp_sport, udp_dport, udp_len, udp_csum
Acid_ag	Ag_id, ag_name, ag_desc, ag_ctime, ag_ltime
Acid_ag_alert	Ag_id, ag_sid
Acid_event	Sid, cid, signature, sig_name, sig_class_id, sig_priority, timestamp, ip_src, ip_dst, ip_proto, layer4_sport, layer 4_dport.
Acid_ip_cache	Ipc_ip, ipc_fqdn, ipc_dns_timestamp, ip_whois,

	ip_whois_timestamp
Data	Sid, cid, data_payload.
Detail	Detail_type, detail_text
Encoding	Encoding_type, encoding_text
Event	Sid, cid, signature, timestamp
Icmphdr	Sid, cid, icm_type, icmp_code, icmp_csum, icmp_id, icmp_seq
Iphdr	Sid, cid, ip_src, ip_dst, ip_ver, ip_hlen, ip_tos, ip_len, ip_id, ip_flags, ip_off, ip_ttl, ip_proto, ip_csum
Opt	Sid, cid, optid, opt_proto, opt_code, opt_len, opt_data

Relaciones entre entidades

Relación	**Entidades participantes**	**Cardinalidad**
Usuario tieneasignado un perfil	Usuario Perfiles	Uno a muchos
Perfiles tiene asignado un rulesets	Perfiles Rulesets	Uno a uno
Persona tiene asignado un Usuario	Usuario Persona	Uno a uno
Acid_ag tiene asignado un	Acid_ag Acid_ag_alert	Uno a uno

acid_ag_alert		
Acid_event tiene asignado muchos acid_ag_alert	Acid_event Acid_ag_alert	Uno a muchos
Acid_ag tiene asignado un acid_ip_cache	Acid_ag Acid_ip_cache	Uno a uno
Base_roles tiene asignados base_users	Base_roles Base_users	Muchos a muchos
Encoding tiene asignado un detail	Encoding detail	Uno a uno
Grupo_perfiles tiene asignado muchos perfiles	Grupo_perfiles Perfiles	Uno a muchos
Sensor tiene asignado muchos icmp_hdr	Icmp_hdr sensor	Uno a muchos
Sensor tiene asignado muchos ip_hdr	Iphdr Acid_event	Uno a muchos
Reference tiene un reference_system	Reference Reference_system	Uno a uno
Grupo_perfiles tiene asignado muchos rulesets	Rulesets Grupo_perfiles	Uno a muchos
Acid_ag_alert tiene asignado un sensor	Acid_ag_alert Sensor	Uno a uno
Sensor tiene asignado muchos acid_event	Acid_event Sensor	Uno a uno
Un sig_class tiene asignado un signature	Sig_class Signature	Uno a uno
Un sig_reference tiene asignado muchos signatures	Signature Sig_reference	Uno a muchos
sensor tiene asignado muchos udphdr	Udphdr sensor	Uno a muchos
sensor tiene asignado muchos tcphdr	Tcphdr sensor	Uno a muchos
Data tiene asignado un sensor	Data Sensor	Uno a uno


persona
id_persona INT(11)
nombres TEXT
apellidos TEXT
ciudad TEXT
cedula VARCHAR(11)
telefono TEXT
fecha_nacimiento DATE
fecha_registro TIMESTAMP
usuario_id_usuario INT(11)
Indexes
usuario
id_usuario INT(11)
login VARCHAR(20)
clave VARCHAR(500)
alias INT(11)
Indexes
data
sid INT(10)
cid INT(10)
data_payload TEXT
Indexes
rulesets
id INT(11)
accion INT(11)
protocolo INT(11)
ip_origen TEXT
puerto_origen TEXT
ip_destino TEXT
puerto_destino TEXT
opciones VARCHAR(2000)
grupo_perfil INT(11)
estado TINYTEXT
direccion VARCHAR(2)
Indexes
acid_ag
ag_id INT(10)
ag_name VARCHAR(40)
ag_desc TEXT
ag_ctime DATETIME
ag_ltime DATETIME
acid_ag_alert_ag_id INT(10)
acid_ag_alert_ag_sid INT(10)
acid_ag_alert_ag_cid INT(10)
Indexes
acid_ag_alert
ag_id INT(10)
ag_sid INT(10)
ag_cid INT(10)
acid_event_sid INT(10)
acid_event_cid INT(10)
Indexes
base_users
usr_id INT(11)
usr_login VARCHAR(25)
usr_pwd VARCHAR(32)
usr_name VARCHAR(75)
role_id INT(11)
usr_enabled INT(11)
usuario_id_usuario INT(11)
Indexes
acid_ip_cache
ipc_ip INT(10)
ipc_fqdn VARCHAR(50)
ipc_dns_timestamp DATETIME
ipc_whois TEXT
ipc_whois_timestamp DATETIME
acid_ag_ag_id INT(10)
acid_ag_acid_ag_alert_ag_id INT(10)
acid_ag_acid_ag_alert_ag_sid INT(10)
acid_ag_acid_ag_alert_ag_cid INT(10)
Indexes
acid_event
sid INT(10)
cid INT(10)
signature INT(10)
sig_name VARCHAR(255)
sig_class_id INT(10)
sig_priority INT(10)
timestamp DATETIME
ip_src INT(10)
ip_dst INT(10)
ip_proto INT(11)
layer4_sport INT(10)
layer4_dport INT(10)
Indexes
perfil
id_perfil INT(11)
nombre TEXT
descripcion TEXT
estado VARCHAR(2)
id_usuario INT(11)
url TEXT
usuario_id_usuario INT(11)
rulesets_id INT(11)
grupo_perfiles_id INT(11)
Indexes
base_roles
role_id INT(11)
role_name VARCHAR(20)
role_desc VARCHAR(75)
base_users_usr_id INT(11)
Indexes
sensor
sid INT(10)
hostname TEXT
interface TEXT
filter TEXT
detail TINYINT(4)
encoding TINYINT(4)
last_cid INT(10)
iphdr_sid INT(10)
iphdr_cid INT(10)
tcphdr_sid INT(10)
tcphdr_cid INT(10)
udphdr_sid INT(10)
udphdr_cid INT(10)
icmphdr_sid INT(10)
icmphdr_cid INT(10)
acid_event_sid INT(10)
acid_event_cid INT(10)
Indexes
icmphdr
sid INT(10)
cid INT(10)
icmp_type TINYINT(3)
icmp_code TINYINT(3)
icmp_csum SMALLINT(5)
icmp_id SMALLINT(5)
icmp_seq SMALLINT(5)
Indexes
signature
sig_id INT(10)
sig_name VARCHAR(255)
sig_class_id INT(10)
sig_priority INT(10)
sig_rev INT(10)
sig_sid INT(10)
sig_gid INT(10)
sig_reference_sig_id INT(10)
sig_reference_ref_seq INT(10)
Indexes
parametros_generales
id INT(11)
tipo_id INT(11)
descripcion VARCHAR(100)
estado CHAR(1)
orden INT(11)
Indexes
grupo_perfiles
id INT(11)
url TEXT
descripcion TINYTEXT
nombre TEXT
id_usuario INT(11)
estado VARCHAR(2)
id_perfil INT(11)
Indexes
reference_system
ref_system_id INT(10)
ref_system_name VARCHAR(20)
Indexes
event
sid INT(10)
cid INT(10)
signature INT(10)
timestamp DATETIME
sensor_sid INT(10)
sensor_iphdr_sid INT(10)
sensor_iphdr_cid INT(10)
sensor_tcphdr_sid INT(10)
sensor_tcphdr_cid INT(10)
sensor_udphdr_sid INT(10)
sensor_udphdr_cid INT(10)
sensor_icmphdr_sid INT(10)
sensor_icmphdr_cid INT(10)
sensor_acid_event_sid INT(10)
sensor_acid_event_cid INT(10)
Indexes
iphdr
sid INT(10)
cid INT(10)
ip_src INT(10)
ip_dst INT(10)
ip_ver TINYINT(3)
ip_hlen TINYINT(3)
ip_tos TINYINT(3)
ip_len SMALLINT(5)
ip_id SMALLINT(5)
ip_flags TINYINT(3)
ip_off SMALLINT(5)
ip_ttl TINYINT(3)
ip_proto TINYINT(3)
ip_csum SMALLINT(5)
Indexes
reference
ref_id INT(10)
ref_system_id INT(10)
ref_tag TEXT
reference_system_ref_system_id INT(10)
Indexes
sig_reference
sig_id INT(10)
ref_seq INT(10)
ref_id INT(10)
Indexes
tcphdr
sid INT(10)
cid INT(10)
tcp_sport SMALLINT(5)
tcp_dport SMALLINT(5)
tcp_seq INT(10)
tcp_ack INT(10)
tcp_off TINYINT(3)
tcp_res TINYINT(3)
tcp_flags TINYINT(3)
tcp_win SMALLINT(5)
tcp_csum SMALLINT(5)
tcp_urp SMALLINT(5)
Indexes
udphdr
sid INT(10)
cid INT(10)
udp_sport SMALLINT(5)
udp_dport SMALLINT(5)
udp_len SMALLINT(5)
udp_csum SMALLINT(5)
Indexes
sig_class
sig_class_id INT(10)
sig_class_name VARCHAR(60)
reference_ref_id INT(10)
reference_reference_system_ref_system_id INT(10)
Indexes

ANEXO C

Script de Recuperación de Fallo

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <pthread.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <signal.h>

#define SLEEP_INTERVAL 200  //miliseconds
#define TIMEOUT_FACTOR 8 //times SLEEP_INTERVAL
#define RECV_TIME_INI 2 //seconds

//#define COMMAND "ip link set br0 down"
#define COMMAND "iptables -F;iptables -A INPUT -i lo -p udp --dport 666 -j NFQUEUE --queue-num
1;iptables -A INPUT -i lo -p udp --dport 666 -j NFQUEUE --queue-num 2 "
//#define RECOVER_COMMAND "ip link set br0 up"
#define RECOVER_COMMAND "iptables -F; /bin/sh /root/firewall"
#define AUTOREC_COMMAND "killall -9 suricata;exec nohup setsid suricata -c /etc/suricata/suricata.yaml -
D -q 1 -q 2 > /dev/null 2>&1 &"
#define AUTOREC2_COMMAND "killall -9 suricata"

int cnt_err=0, panic=0;
sigset_t block_sig;

void * funca_th(void *);
//void exec_recov(int sig);

int main(int argc, char **argv){
        int sd_out = socket(PF_INET,SOCK_DGRAM,0);
        char *msg="123456789012345678901 2";
        struct sockaddr_in s_out, s_temp;
        pthread_t th0;
        struct timespec *interval=malloc(sizeof(struct timespec));
        int rc, auto_rec=0;
        //int ch_pid=0;
        int recv_time = RECV_TIME_INI;

        //printf("starting");
        sigemptyset (&block_sig);
        sigaddset (&block_sig, SIGUSR1);

        if(argc==2 && !strcmp(argv[1],"auto")){
```

```
                auto_rec=1;
        }


        pthread_create(&th0, NULL, &funca_th, NULL);

        interval->tv_sec = (SLEEP_INTERVAL/1000);

        if(interval->tv_sec)
                interval->tv_nsec = SLEEP_INTERVAL * 1000000 - (long)interval->tv_sec * 1000000000;
        else
                interval->tv_nsec = SLEEP_INTERVAL * 1000000;

        if(sd_out == -1){
                //printf("no sock\n");
                return 1;
        }

        s_out.sin_family = AF_INET;
        s_out.sin_port = htons(666);
        inet_aton("127.0.0.1",&(s_out.sin_addr));

        s_temp.sin_family = AF_INET;
        s_temp.sin_port = 0;
        inet_aton("127.0.0.1",&(s_temp.sin_addr));

        rc = bind (sd_out, (struct sockaddr *)&s_temp, sizeof(struct sockaddr_in));

        sleep(1);

        while(1){
                //printf("sending\n");
                rc = sendto(sd_out,msg,strlen(msg),0,(struct sockaddr *)&s_out,sizeof(struct
sockaddr_in));
                nanosleep(interval,NULL);

                if(cnt_err <= TIMEOUT_FACTOR + 1)
                        cnt_err++;

                if(rc==-1){
                        printf("%s\n",strerror(rc));
                }

                //printf("cnt_err %d",cnt_err);

                //if you recover from panic!!!
                if(panic && cnt_err != TIMEOUT_FACTOR + 2){
                        //printf("recovering 1 \n");
                        system(RECOVER_COMMAND);
                        //printf("recovered\n");
                        panic=0;
                        recv_time = RECV_TIME_INI;
                }
                //try to auto recover
```

```
                else if(panic && cnt_err == TIMEOUT_FACTOR + 2 && auto_rec){
                        //printf("recovering\n");
                        sleep(recv_time);
                        recv_time++;
                        panic=0;
                        system(AUTOREC_COMMAND);
                }

                if(cnt_err == TIMEOUT_FACTOR + 1){
                        //printf("panic!!!\n");
                        system(COMMAND);
                        panic=1;
                }

        }
        return 0;
}


void * funca_th(void * vv){
        struct sockaddr_in s_in;
        int sd_in = socket(PF_INET,SOCK_DGRAM,0);
        char buf[10];

        if(sd_in == -1){
                //printf("no sock\n");
                return NULL;
        }


        s_in.sin_family = AF_INET;
        s_in.sin_port = htons(666);
        inet_aton("127.0.0.1",&(s_in.sin_addr));

        bind (sd_in, (struct sockaddr *)&s_in, sizeof(struct sockaddr_in));

        while(1){
                //printf("wait\n");
                recvfrom(sd_in, buf, 9, 0, NULL, 0);
                cnt_err=0;
        }
}
```

BIBLIOGRAFÍA

[1] **BEALE, J, BAKER, A., CASWELL, B. POOR, M.** "SNORT 2.1: INTRUSION DETECTION", SYNGRESS, 2000.

[2] **KIRDA, E. JHA, S. BALZAROTTI, D.** "RECENT ADVANCES IN INTRUSION DETECTION", SYMPOSIUM ON RECENT ADVANCES IN INTRUSION DETECION, 2009.

[3] **YOUNG, G. PESCATORE, J.** "MAGIC QUADRANT FOR NETWORK INTRUSION PREVENTION SYSTEMS", GARTNER RAS CORE RESEARCH, 2010

[4] **MCREE, R**. "SURICATA IN TOOLSMITH: MEET THE MEERKAT", http://holisticinfosec.blogspot.com/2010/08/suricata-in-toolsmith-meet-meerkat.html, 2010.

[5] **SCARFONE, K. MEL, P,** "GUIDE TO INTRUSION DETECTION AND PREVENTION SYSTEMS", NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY GAITHERSBURG, 2007

[6] **KOOLSTRA, A.** “SURICATA YAML DOCUMENTATION”, https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml, 2010

[7] **GERBER, J.** “SURICATA: A NEXT GENERATION IDS/IPS ENGINE”, http://blog.securitymonks.com/2010/01/05/suricata-a-next-generation-idsips-engine/, 2010

[8] **ALONSO, G.** “WEB SERVICES: CONCEPTS, ARCHITECTURES AND APPLICATIONS”, SPRINGER-VERLAG BERLIN HEIDELBERG, 2004

[9] **FREDERIC, P. AGNES, F. MCBREWSTER** ,**J**. “MODEL-VIEW-CONTROLLER”, VDM PUBLISHING HOUSE LTD, 2010

[10] **STREICHER, M. “**PACKAGING SOFTWARE WITH RPM, PART 1”, http://www.ibm.com/developerworks/library/l-rpm1/, 2010.

[11] **FOSTER, E.** "RED HAT RPM GUIDE", WILEY, 2003

[12] **DENNING**, “AN INTRUSION DETECTION MODEL”, IEEE, 1987.

[13] **AXELSSON, S. “**THE BASE-RATE FALLACY AND THE DIFFICULTY OF INTRUSION DETECTION SYSTEM”, TISSEC, 2000.

[14] **W. LU, H. TONG.** “DETECTING NETWORK ANOMALIES USING CUSUM AND EM CLUSTERING”, SPRINGER-VERLAG BERLIN HEIDELBERG, 2009.

[15] **BINKLEY, J.** “Ourmon – Network Monitoring and Anomaly detection System”, http://ww.ourmon.sourceforge.net, 2005.

[16] **BINKLEY, J. MCHUGH, J. GATES, C**. "LOCALITY, NETWORK CONTROL AND ANOMALY DETECTION", PORTLAND STATE UNIVERSITY - COMPUTER SCIENCE TECHNICAL REPORT, 2009

[17] **BINKLEY, J. PAREKH, D. "**TRAFFIC ANALYSIS OF UDP BASED FLOWS IN OURMON", PORTLAND STATE UNIVERSITY - COMPUTER SCIENCE TECHNICAL REPORT, 2007.

[18] **SMITH, B.**, "TOMAHAWK MANUAL PAGE", http://tomahawk.sourceforge.net/, 2004.

[19] **SANFILIPPO, S.,** "HPING3", http://www.hping.org/hping3.html, 2006.

[20] **BULLOCK, T.** "HTTPERF DOCUMENTATION", http://www.hpl.hp.com/research/linux/httperf/docs.php, 1998.

[21] **NLANR/DST**, "IPERF", http://iperf.sourceforge.net/, 2008