UNIVERSIDAD NACIONAL DE LA PLATA
Facultad de Informática

Integración de herramientas de seguridad para redes informáticas

Tesis presentada para optar al título de Licenciado en Informática

Einar Lanfranco
Matías Pagano

Director de tesis: Lic. F. Javier Diaz
Co-director: Lic. Paula Venosa

Lugar de trabajo: Laboratorio de Investigación en Nuevas Tecnologías Informáticas

La Plata, 25 de octubre de 2010.

LICENCIA

Agradecimientos

Bueno, por fin estamos en esta sección, que si bien es una de las primeras que se ven según el orden del índice de capítulos, es en realidad la última que nos queda, la que nos falta para terminar con esta tesina, y la verdad que pasaron tantos años, con tantos recuerdos, alegres y de los otros, que se hace muy dificil de completar, pero bueno, aquí vamos con el intento.

Gracias a todas aquellas personas que incondicionalmente nos acompañaron en la etapa de nuestras vidas que se cierra con este trabajo: a nuestros papas y mamás, a nuestros hermanos, a nuestras mujeres, Laura y Emilia, a nuestros hijos, a la abuela China, a nuestros tíos y primos, a nuestros amigos, nuestros compañeros de trabajo y a los que estudiaron con nosotros durante todos estos años.

Gracias a Paula, Nico y Javier, sin la influencia de los cuales nunca nos hubiéramos acercado y mucho menos interesado y formado en temas de seguridad.

Gracias a la comunidad del software libre, ya que sin su aporte, generalmente desinteresado, no hubiéramos podidos desarrollar este trabajo ya que no existiría Latex, Lihuen GNU/Linux, Debian GNU/Linux, Snort, Barnyard, Iptables, MySQL, Apache, Catalyst, Subversion, OpenVas, Perl, etc, etc, etc.....y tantos otros productos en los que nos apoyamos para el desarrollo que hicimos.

Si el lector se busca es probable que se encuentre en varios de los renglones de agradecimiento. :D

Índice general

Índice de figuras

Capítulo 1

Objetivo

El objetivo del trabajo es analizar y estudiar los diversos tipos de herramientas de seguridad en redes existentes en la actualidad, a fin de seleccionar un subconjunto de ellas que sean integrables para conformar una única aplicación. El subconjunto debe incluir Firewall, IDS y detector de vulnerabilidades.

El proceso de integración tiene dos objetivos principales:

- Integración propiamente dicha entre las herramientas estudiadas, logrando que las mismas interactúen adecuadamente.
- Desarrollo de un prototipo que implementa un mecanismo único de administración de las distintas herramientas.

La red objetivo es una red del tipo DMZ y las herramientas a utilizar deben ser al menos de uso gratuito, con preferencia a seleccionar herramientas de Software Libre.

1.1. Motivación

El crecimiento exponencial de Internet genera que hoy en día la cantidad de servicios prestados en una red y el acceso a los mismos sea cada vez mayor, aumentando tanto la cantidad de usuarios que acceden como la cantidad de veces que un usuario lo hace, con esta tendencia creciente se genera en los usuarios una dependencia por estos servicios. Para graficar la idea basta recordar la necesidad que uno tenía de acceder al correo electrónico hace algunos años cuando no era de uso masivo, con la necesidad que uno tiene hoy en día donde la mayor parte de la gente utiliza este medio, tanto para temas laborales como para temas personales, como puede ser a través de la utilización de redes sociales como Facebook [1] por ejemplo, tan en auge hoy en día.

Ahora bien, los accesos a los servicios podemos clasificarlos según la intención, hay accesos bienintencionados, de parte de la gente que simplemente es usuario de los sistemas y hay accesos malintencionados, que son aquéllos en los que alguien trata de acceder a datos no autorizados o intenta generar problemas de funcionamiento.

A su vez, el crecimiento mencionado, genera que la cantidad de software que se utiliza haya aumentado drásticamente, trayendo aparejado un aumento de vulnerabilidades y generando una necesidad de monitoreo contínuo del software instalado.

[1] Según el sitio oficial (URL: http://www.facebook.com): Facebook es una herramienta que implementa una red social que pone en contacto a personas con sus amigos y otras personas que trabajan, estudian y viven en su entorno.

Es deseable que una vez que se detecta alguna vulnerabilidad se pueda corregir en el menor tiempo posible o al menos, si no existe una solución para ella, intentar atenuar su impacto tomando las medidas necesarias.

Muchas veces no existen avisos sobre las vulnerabilidades que aparecen o no se puede distinguir fácilmente un acceso malintencionado de otro que no lo es, por eso es necesario tener alguna herramienta que estudie el comportamiento del tráfico en la red para poder reaccionar oportunamente.

Por estos motivos los responsables de la administración de una red deben redoblar sus esfuerzos para poder mantener la misma en el mejor estado de seguridad posible día a día.[1]

Para lograrlo se utilizan habitualmente distintas herramientas, las cuales generan grandes cantidades de información ocasionando que sea dificultoso monitorearlas. Además lograr que dichas herramientas trabajen de manera conjunta, o sea que lo que una herramienta detecta le sirva a alguna de las otras para tomar una medida específica es aún más difícil.

Otro punto que tuvimos en cuenta, además de la necesidad de monitoreo es la necesidad de automatización de algunas tareas, por ejemplo el análisis de vulnerabilidades debe ser periódico, ya que si esta tarea no es automática es altamente probable que dependa de las ganas y el tiempo disponible por el o los responsables de la red.

Ofrecer una solución para las problemáticas aquí planteadas es lo que motivó este trabajo.

1.2. Solución

Ya definido el objetivo lo siguiente que hicimos fue investigar y elegir un subconjunto de herramientas de seguridad, incluyendo una solución de firewall, una solución de IDS[2] y una solución de analizador de vulnerabilidades.

En la elección de todas ellas tratamos de orientar la búsqueda a herramientas de amplia aceptación por la comunidad y los profesionales de la seguridad informática. Tratando de que las mismas estén dentro del universo de las herramientas gratuitas, y si es posible, que sean Software Libre.

Teniendo en cuenta que nosotros desde algún tiempo y actualmente trabajamos con software libre, particularmente con software GNU[3] y con Linux[4][5], tanto en los desarrollos que hacemos como en las cátedras de la Facultad en las que participamos, tomamos la experiencia personal como un factor de peso a la hora de la elección.

Una vez realizada la elección desarrollamos un prototipo de una aplicación web que permite manejarlas de forma centralizada. Teniendo como premisa a la hora del desarrollo que la solución construida sea extendida permitiendo la incorporación de otras herramientas al prototipo original.

1.3. Estructura organizativa del trabajo

En este trabajo se presentan diversos conceptos relacionados con herramientas de seguridad, principalmente firewalls, detectores de intrusiones y analizadores de vulnerabilidades. Se brindan los detalles necesarios para comprender cómo funcionan estas aplicaciones y se presenta un análisis de las implementaciones existentes en el mercado, principalmente de aquéllas que son distribuídas como software libre. Luego

de presentarlas se relata el proceso de selección de un subconjunto de ellas, utilizando diversos criterios y se justifica la selección en cada caso.

Una vez detallado el proceso de selección se describe el trabajo realizado para integrarlas en un sistema web único remarcando las funcionalidades más importantes disponibles en el sistema.

Además se presenta un resumen del esquema interno de la aplicación relatando desde aspectos como la metodología de programación utilizada y el esquema de la base de datos resultante, hasta una descripción de los estándares que nuestra aplicación respeta.

En la parte final de este documento se incluye una lista de las ideas que tenemos para el futuro de nuestra aplicación y las conclusiones resultantes del presente estudio y desarrollo.

Capítulo 2

Herramientas de seguridad a integrar

2.1. FW

Un firewall es un elemento de hardware o software que se utiliza en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido el o los responsables de la red. La ubicación habitual de un firewall es el punto de conexión de la red interna de la organización con la red exterior, que normalmente es Internet; de este modo se protege la red interna de intentos de acceso no autorizados desde Internet, que puedan aprovechar vulnerabilidades de los sistemas de la red interna. Ver figura 2.1.

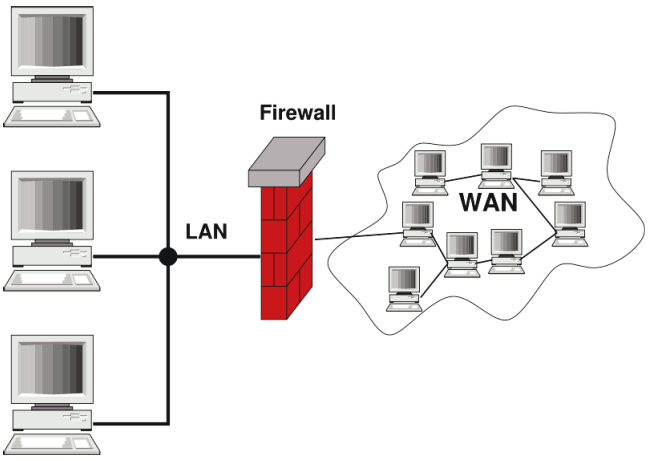

Figura 2.1: Firewall

2.1.1. Características

Su modo de funcionar es indicado por la recomendación RFC 2979[6]: Los Firewalls pueden actuar tanto como un punto final o relay de un protocolo (por ej., un cliente/servidor SMTP o un Web proxy agent), como un filtro de paquetes, o como una combinación de ambos.

Cuando actúa como punto final de un protocolo puede:

1. implementar un subconjunto “seguro” del protocolo,

2. realizar chequeos de validez adicionales del protocolo,

3. utilizar una metodología de implementación diseñada para minimizar la probabilidad de bugs,

4. correr en un ambiente aislado y "seguro" o

5. utilizar alguna combinación de las técnicas mencionadas en tándem.

Cuando el firewall actúa como un filtro de paquetes, éste examina cada paquete y

1. pasa el paquete hacia el otro lado sin modificarlo,

2. elimina el paquete entero, o

3. manipula el paquete en sí de alguna manera.

Típicamente los firewalls basan sus decisiones en las direcciones IP origen y destino, en los números de puerto origen y destino, y en los flags de los paquetes.

Las aplicaciones deben continuar trabajando apropiadamente ante la presencia de un firewall. Esto se traduce en la siguiente regla de transparencia:

La introducción de un firewall NO DEBE causar fallas no deseadas ante un uso legítimo y que cumple con los estándares, el cual funcionaría si éste no estuviera presente.

Notar que este requerimiento sólo se aplica ante una conexión legítima, un firewall tiene derecho a bloquear cualquier acceso que considere ilegítimo, independientemente de si el intento de acceso es o no compatible con los estándares. Esta es, después de todo, la razón de utilizar un firewall en primer lugar.

2.1.2. Tipos de Firewall

Existen diferentes tipos de firewalls. La clasificación se basa en el mecanismo que utilizan para permitir o prohibir el tráfico de red. Estos son:

1. Firewall de capa de red o de filtrado de paquetes (Paquet filtering)

2. Firewall de capa de aplicación (Stateful Firewall)

3. Proxy Firewall

4. Firewall personal

Firewall de capa de red o de filtrado de paquetes (Paquet filtering)

Funciona a nivel de red (capa 3 del modelo OSI[7] y capa 3 del stack de protocolos TCP-IP[8]) como filtro de paquetes IP[9]. A este nivel se pueden realizar filtros según los distintos campos de los paquetes IP: dirección IP origen, dirección IP destino. A menudo en este tipo de firewalls se permiten filtrados según campos de nivel de transporte (capa 4 de TCP-IP y Modelo OSI), como el puerto origen y destino, o a nivel de enlace de datos (Capa 2 de ambos Modelos) como la dirección MAC[10].

Hay que aclarar que hay bibliografía que se referencia a el modelo TCP-IP como un modelo de 4 capas en lugar de 5 como las indicadas en la comparación de la figura 2.2.. Nosotros nos basamos en el esquema adoptado en el libro Redes de Computadoras: Un enfoque descendente[11].

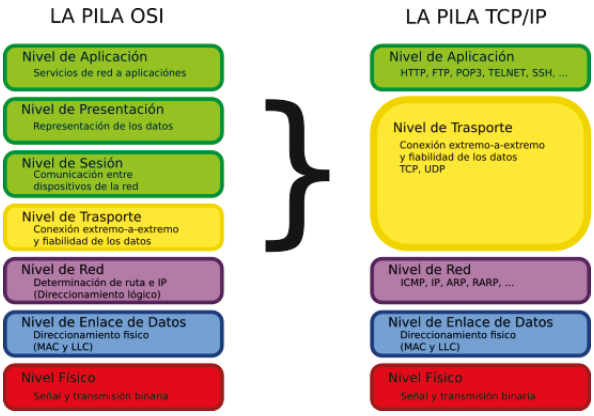

Figura 2.2: Comparación ISO y TCP/IP

Firewall de capa de aplicación (Stateful Firewall)

Trabaja en el nivel de aplicación (nivel 7), de manera que los filtrados se pueden adaptar a características propias de los protocolos de este nivel. Por ejemplo, si se trata de tráfico HTTP[12], se pueden realizar filtrados según la URL a la que se está intentando acceder. Tiene en cuenta, principalmente, información de capa de transporte y de aplicación del paquete que inicia la conexión. Si el firewall permite pasar el primer paquete, se agrega una entrada en la tabla de estados y los paquetes pertenecientes a dicha conexión se permiten sin ser analizados.

Proxy Firewall

El proxy permite una conexión indirecta desde y hacia Internet. Cada comunicación cliente-servidor requiere dos conexiones: una desde el cliente al firewall, el cual actúa como "proxy" del servidor deseado, y otra desde el firewall hacia el servidor deseado.

Firewall personal

Es un caso particular de firewall que se instala como una aplicación en un host, filtrando las comunicaciones entre dicho host y el resto de la red y viceversa.

2.2. IDS[2]

Podemos definir la detección de intrusiones como la tarea de detectar actividad sospechosa, incorrecta o inapropiada, que permite identificar ataques, incidentes de seguridad y errores de configuración.

2.2.1. Características

Los IDS o sistemas de detección de intrusiones son aplicaciones en ejecución constante que están habitualmente diseñadas para examinar tráfico de red y/o archivos de logs en busca de patrones que indiquen comportamiento sospechoso. Una vez que este tipo de actividad es descubierta, la misma aplicación se encarga de identificarlo y de alertar sobre su presencia. Mediante el análisis de estas alertas tenemos la posibilidad de tomar las medidas necesarias para contrarrestar la actividad maliciosa o corregir los errores de configuración que se hayan detectado.

La pregunta que suele surgir es: *¿Si ya se tiene un firewall, para qué se quiere un IDS?* Un sistema de detección de intrusiones funciona a manera de complemento del firewall, es un mecanismo de control que nos sirve para monitorear qué pasa dentro de nuestro entorno, una vez que se superó la barrera impuesta por el firewall.

Refiriéndonos a un ejemplo de la vida real actual, podemos decir que un firewall sería lo equivalente a tener una cerca con una caseta de seguridad en uno de los hoy difundidos barrios privados en donde el guardia que esta allí decide quien puede y quien no puede entrar y salir. Ahora bien, cuando alguna persona logra superar esa barrera, el guardia de la entrada que lo dejó pasar, no controla cómo se porta el visitante dentro del barrio privado; para ésto tendríamos que tener otro mecanismo, ya sean cámaras y/o más guardias de seguridad, que controlen el comportamiento dentro del barrio. Estas cámaras y guardias extras serían el equivalente al IDS.

2.2.2. Tipos de IDS

Si bien hay distintas clasificaciones cuando se lee acerca de IDS, vamos a presentar en primer instancia una que los agrupa según qué es lo que el IDS analiza en busca de comportamiento anormal.

Basados en HOST

Un IDS basado en host monitorea los logs del mismo con el fin de detectar actividad sospechosa, en general es configurable para elegir qué logs del sistema queremos monitorear. Puede detectar por ejemplo la instalación de una nueva aplicación o un intento de inicio de sesión fallido.

Nos brinda mayor poder de análisis ya que podríamos por ejemplo reensamblar tráfico segmentado y analizarlo cuando está completo para ver si el contenido es realmente algo peligroso, procedimiento que no se puede hacer en un IDS de red, ya que en vez de reenviar el tráfico deberíamos almacenarlo para juntar las partes y analizarlo.

Basados en Red

Un IDS basado en red monitorea todo el tráfico que llega a la placa de red en la que está escuchando, reaccionando ante cualquier anomalía o signo de actividad sospechosa. Si la aplicación está en un equipo conectado a un switch como nodo final en la red, va a procesar todo el tráfico que llega a esa máquina solamente, pero si está en un lugar por donde pasa todo el tráfico de la red, como puede ser en un router de borde o en un puerto de monitoreo del switch, podrá analizar todo el tráfico de la red. Podemos decir que funciona como un sniffer[1] que busca patrones anormales en los paquetes observados.

Basados en filesystem

Un IDS basado en filesystem monitorea cambios dentro del sistema de archivos de un host. Al vulnerar un sistema habitualmente los atacantes alteran algunos archivos, tanto para cubrir sus rastros y evitar que se sepa de su ingreso, como para asegurarse la posibilidad de acceder nuevamente al equipo en un futuro. Para poder monitorear ésto existen los IDS basados en filesystem, que lo que hacen es sacar una foto del sistema de archivos, aplicando una función de hash sobre los archivos, que

[1] Un sniffer es una aplicación que permite capturar el tráfico de una red, para así poder analizarlo.

luego compara con nuevas fotos de manera periódica para determinar si algo fue cambiado.

Evolución en los métodos de funcionamiento de los IDS

Esta segunda clasificación que presentamos es según la respuesta que pueden dar los sistemas ante la detección de un incidente.

IDS

En su etapa inicial los IDS funcionaban generando archivos de logs y alertas para que leyera el usuario, generalmente el administrador del sistema, el cual era el encargado de reaccionar ante el aviso generado. Si bien estas alertas fueron evolucionando desde simples archivos de texto plano, que el usuario debía buscar y leer, a datos almacenados en una base de datos y a envíos de las alertas por mail y/o por SMS, en este esquema es el usuario el que debe tomar alguna acción para contrarrestar o solucionar las anomalías detectadas.

En resumen, el IDS no evitaba ni contrarrestaba el evento, simplemente lo detectaba y avisaba al usuario. Para graficarlo podemos citar el siguiente ejemplo: al detectar un escaneo de puertos desde Internet a nuestro servidor web lo que el IDS hacía era, en el mejor de los casos, enviar un mail al administrador del sitio alertando respecto de la ocurrencia de dicho escaneo, en cuyo caso éste terminaba con resultados positivos para el atacante y nada se hacía para impedir que el ataque ocurriera. O sea el atacante era detectado, pero no se hace nada para impedir o minimizar los efectos del ataque.

IDS con respuesta activa

Lo siguiente que ocurrió en el mercado fue la evolución de las técnicas de respuesta, primero aparecieron aplicaciones diseñadas para acoplarse a los IDS existentes, su funcionamiento consiste en monitorear los logs de los IDS y tomar alguna acción ante la aparición de determinado evento, incluyéndose esa funcionalidad luego directamente en los detectores de intrusiones.

Entonces podemos decir que los IDS agregaron a su funcionalidad, además de continuar con la generación de alertas, funciones de respuesta activa, de esta manera podemos, en el ejemplo citado antes, cuando ocurre un escaneo de puertos a nuestro servidor web, el IDS lo detecta y genera una respuesta que podría ser comunicarse con el firewall del equipo y provocar que bloquee la IP origen del escaneo, evitando que pudieran ocurrir nuevas conexiones desde la IP filtrada.

La ventaja de esta metodología es que además de ser detectado, se evitan nuevos ataques dejando, desde el momento de la detección, nuestra red protegida hacia el futuro. Como desventaja podemos ver por un lado, que nada se hace para evitar el primer ataque, sino que las contra-medidas se toman una vez que el ataque se consumó, y por otro lado que podemos caer fácilmente en problemas de disponibilidad: el atacante podría simular tener otra dirección IP, por ejemplo la de un servidor de mail, entonces nuestro sistema filtraría el tráfico desde esa dirección IP fingida, causando que desde nuestra red sea imposible acceder a esa dirección IP, generando una denegación de servicio.

IPS

Finalmente los sistema evolucionaron desde sistema de detección de intrusiones a sistemas de prevención de intrusiones, es decir que lo que se hace actualmente es analizar el tráfico antes de dejarlo pasar hacia el destino final, con esta metodología evitamos que el tráfico siquiera llegue a nuestro servidor si es que lo catalogamos como tráfico anómalo.

Por ejemplo, en el escaneo mencionado antes, al llegar el tráfico a nuestro sistema el firewall lo pasa directamente al IDS que lo analiza, y de acuerdo al resultado del análisis puede bloquearlo, modificar el contenido del paquete o dejarlo pasar hacia la aplicación a la que está dirigido. La ventaja es que de esta forma evitamos que el tráfico maligno llegue siquiera una vez a nuestro aplicativo y que se consiga realmente concretar algún ataque a la disponibilidad del servicio ya que el tráfico se analiza cada vez que pasa. Como desventaja podemos decir que este tipo de implementación generalmente requiere más recursos de procesador que los anteriores. Esto se debe a que al estar en el medio de todo, se vuelve indispensable que el IDS esté en ejecución para permitir el paso del tráfico. Si se cae dejará de pasar tráfico, afectando la disponibilidad de la red.

Finalmente vale aclarar que muchos de los sistemas funcionan de manera híbrida, combinando técnicas, y los productos en general pueden configurarse de cualquiera de estas formas.

Otra clasificación para los IDS basados en red

Como última referencia queremos citar una nueva clasificación que nos parece de suma importancia, clasificarlos según el tipo de tráfico que pueden analizar.

IDS que funcionan en tiempo real

Este tipo de IDS analiza el tráfico en tiempo real, a medida que éste pasa por el punto donde está escuchando.

IDS con capacidad de análisis de captura de tráfico

Estos IDS basados en red tienen la capacidad de analizar el tráfico capturado y almacenado en un momento posterior al paso del tráfico por el sensor.

El análisis posterior se considera de suma importancia ya que como se suele decir "la seguridad es un proceso, no un estado", lo que quiere decir que lo que hoy es seguro probablemente mañana no lo sea, de hecho lo que en este momento es seguro en un rato es bastante probable que deje de serlo o incluso algo que creemos es seguro es posible que ya no lo sea. Con esto queremos decir que lo que hoy el IDS cataloga como tráfico normal es probable que dentro de algún tiempo sea catalogado como tráfico ofensivo.

Mediante este tipo de análisis tenemos, por ejemplo, la posibilidad de poder reconstruir el comportamiento de un virus que nos infectó ayer sin ser detectado y remediar los problemas que causó, ya que podremos estudiar su comportamiento y ver por ejemplo con qué sistemas se conectó.

En general los IDS que soportan el análisis diferido soportan también el análisis en tiempo real.

2.3. Analizadores de Vulnerabilidades

Hay diversos tipos de analizadores existentes en el mercado y podríamos establecer muchas categorizaciones distintas para estas herramientas, en este trabajo vamos a presentar una clasificación de acuerdo al propósito que se persigue con cada una de estas utilidades, describiendo brevemente en cada caso las técnicas utilizadas para obtener los resultados buscados.

2.3.1. Características

Un escaneador de vulnerabilidades[13] es un programa diseñado para buscar fallas o debilidades en las aplicaciones o en su configuración. Típicamente el ambiente de búsqueda es un host, un grupo de hosts en particular o todos los hosts de una red o subred dada.

Si bien varía de acuerdo a la aplicación que se utilice y a la forma en que se lo configure, habitualmente un escaneador primero busca direcciones IP activas, luego trata de identificar los puertos abiertos de cada una de esos hosts activos y finalmente trata de identificar qué sistema operativo y/o aplicaciones se están ejecutando en cada uno de ellos.

Una vez que está identificado todo lo que corre en cada equipo el escaneador intentará averiguar la versión exacta de cada aplicación y cada sistema operativo para averiguar el nivel de actualización. Luego del análisis, generalmente presentan un informe al usuario con los resultados obtenidos, los que muchas veces tienen referencias a documentación que nos permite entender los problemas descubiertos.

Más allá del hasta ahora descripto funcionamiento típico de un escaneador de vulnerabilidades, hay muchas alternativas de la implementación específica, y existen algunos de ellos que tienen propósitos de ataque donde adicionalmente al descubrimiento de los bugs[2] se incluye el exploit[3], e incluso el payload[4] para poder sacar provecho de las vulnerabilidades descubiertas.

2.3.2. Tipos de Analizadores

Escaneador de Puertos

Antes de hablar de escaneadores de puertos vamos a tener que realizar una pequeña referencia al modelo TCP/IP.

Hoy en día el stack más difundido en Internet es el del protocolo TCP/IP, por eso es que este es el lenguaje común que hay que entender para poder estudiar y trabajar en el mundo de las redes. Para poder comprender como funcionan los escaneadores de puertos tenemos que comprender que los servicios que se prestan en Internet están referenciados por dos elementos: una dirección de red del host donde reside el servicio y un número de puerto que lo identifica.

No vamos a entrar en el mundo que significaría explicar como funcionan ni las direcciones de red ni la forma en que se las maneja, simplemente diremos que con

[2]Un bug es el resultado de un fallo o deficiencia durante el proceso de creación de un sistema operativo y/o aplicación.

[3]Un exploit es una pieza de software, un fragmento de datos, o una secuencia de comandos con el fin de automatizar el aprovechamiento de un error, fallo o vulnerabilidad, a fin de causar un comportamiento no deseado o imprevisto en los programas informáticos, hardware, o componente electrónico.

[4]El payload es el código que un exploit transmite y ejecuta en el host vulnerable.

la dirección podríamos llegar a identificar un host unívocamente en la red a la que está conectado, y a veces hasta encontrarlo en Internet, de acuerdo al tipo de dirección que tenga asignada, pública o privada.

En cuanto a los puertos si nos vamos a adentrar un poco más, para cada uno de los protocolos principales, **TCP**[5] y **UDP**[6], hay 65536 identificadores de puertos distintos y utilizables, del puerto 0 al puerto 65535 para cada uno.

Lo que uno hace cuando escribe un servicio que atenderá a los clientes en la red es ponerlo a escuchar en un puerto en particular a la espera de requerimientos que llegarán allí. La mayoría de los servicios existentes usan un número de puerto dentro de un rango limitado, esto es así para simplificar y permitir el uso. Por ejemplo, es muy común, que un servidor web utilice puertos como el TCP 80 o el TCP 443, por eso cuando uno usa un navegador web y trata de acceder a un servidor éste intentará llegar al puerto 80 si usa HTTP y al 443 si usa HTTPS, estableciéndose de este modo la conexión, ya que si el servidor usara un puerto aleatorio sería más complicado para el navegador encontrarlo. El usuario debería especificar el puerto como parte de la URL.

Si bien para estandarizar el uso de los puertos, cuando un servicio se hace muy común, hay un organismo, el IANA[16] que le asigna el puerto a ese servicio, usándolo luego como referencia, cualquiera puede desarrollar una aplicación y ponerla a escuchar en cualquier puerto, siempre que este puerto no esté siendo usado para otra aplicación corriendo en ese equipo. Esta asignación estandarizada servirá también de referencia para el atacante, ya que si al hacer un escaneo encuentra un puerto abierto puede a partir del número y protocolo del puerto intuir qué tipo de software es el que está ahí escuchando.

Definición

Habiendo introducido ya el concepto de puerto, un escaneador de puertos es una pieza de software que realiza pruebas en uno o más hosts en busca de puertos abiertos, es decir, en puertos donde hay servicios escuchando.

El uso de estas herramientas tiene usualmente dos enfoques, los administradores lo utilizan para chequear el estado de sus redes, por ejemplo para saber si hay más puertos abiertos de los que debería haber en un servidor, y, los atacantes, que lo usan para construir un perfil de la red y saber a qué puertos de que equipos pueden atacar.

De acuerdo a la forma en que se usa suele dársele dos denominaciones distintas, si estamos escaneando un host para saber qué puertos tiene abierto estamos hablando de un *portscan*, y si estamos buscando un puerto en particular entre varios host estamos haciendo un *portsweep*.

Ahora bien, cuando pasamos un escaneador de puertos podemos encontrar que la respuesta encuadra en alguno de los siguientes estados posibles:

- **Abierto**: es cuando el escaneador interpreta que hay algún servicio que se presta en ese puerto.
- **Cerrado**: es cuando el escaneador interpreta que no hay ningún servicio que se presta en ese puerto.

[5]TCP (Transmission Control Protocol):protocolo de comunicación orientado a conexión y fiable del nivel de transporte. **RFC 793** [14]

[6]UDP (User Datagram Protocol): protocolo de comunicación no orientado a conexión. **RFC 768**[15]

- **Filtrado o Bloqueado**: es cuando el escaneador interpreta que hay algún servicio que se presta en ese puerto pero que esta siendo filtrado por ejemplo por un firewall.

¿Cómo sabe el estado del puerto?

En general podemos decir que para saber el estado de un puerto utilizamos distintas técnicas, que consisten básicamente en manipular el protocolo TCP o UDP, de acuerdo al tipo de puerto que buscamos testear. Con algunas de ellas lo que se hace es un uso normal del protocolo, y en otras se utilizan segmentos o datagramas que los protocolos consideran inválidos, y así, mediante el análisis de las respuestas que el objetivo genera ante estos eventos podemos clasificar el estado del puerto.

Escaneando puertos

Vamos a relatar la metodología utilizada para puertos TCP primero y UDP luego. Para comprender estos mecanismos necesitaremos entender como funcionan los protocolos.

Funcionamiento de una conexión utilizando TCP

TCP es un protocolo orientado a conexiones, por lo tanto, para que una aplicación pueda intercambiar datos por TCP, previamente debe establecerse una conexión entre el cliente y el servidor. Este establecimiento está definido en la RFC 793 y habitualmente es conocido como el saludo de tres vías o three way handshaking.

Lo que ocurre primero es que el cliente inicia el pedido de establecimiento de la conexión enviando al servidor un segmento TCP con el flag SYN activado, el servidor contesta confirmando la recepción de este segmento TCP con un nuevo segmento con los flags SYN/ACK activados a lo que el cliente responde con un nuevo segmento con el flag ACK activado completando así el establecimiento de la conexión. A partir de ese momento la aplicación puede empezar a intercambiar datos propios a través del canal establecido.

Una vez que la aplicación termina de intercambiar los datos, la conexión se cerrará con un intercambio de segmentos FIN y ACK en ambos sentidos. Ver figura 2.3.

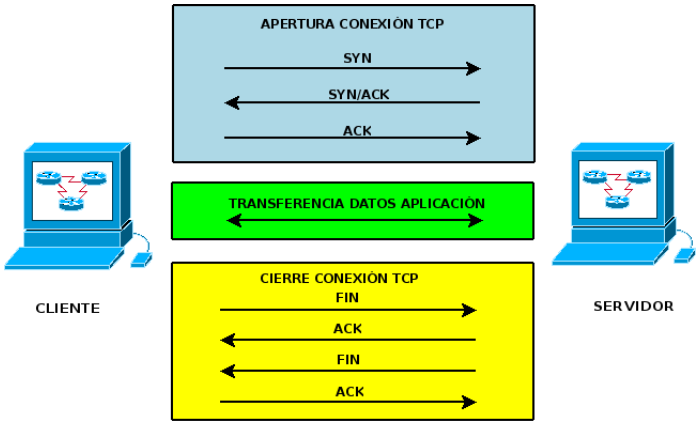

Figura 2.3: Conexión TCP

.

Comprendiendo cómo funciona lo anterior, pasamos a describir las distintas técnicas que existen.

Técnicas de scanning para puertos TCP

Si bien hay distintas formas de clasificar las técnicas, presentaremos en este caso la que nos parece más adecuada. Las categorías aquí presentadas van desde técnicas simples que dependen de la forma en que se manipulen segmentos TCP hasta técnicas más complejas donde intervendrán distintas herramientas para poder llevar a cabo el escaneo, todas ellas con distintas características.

- Open TCP scanning:
  - TCP Connect() Scanning (Vanilla connect scanning)
- Stealth TCP scanning
  - Half-open SYN flag scanning
  - Inverse TCP flag scanning
  - ACK flag probe scanning
- Third-party and spoofed TCP scanning
  - TCP Idle Scanning (IP ID header scanning)
  - Proxy bounce scanning
  - Sniffer-based spoofed scanning

Open TCP scanning

TCP Connect() Scanning (Vanilla connect scanning)

Esta técnica consiste simplemente en intentar abrir una conexión TCP, si se logra, inmediatamente establecida se cerrará, enviando un RST y se considerará el puerto como abierto. Si el puerto está cerrado el objetivo responderá al SYN del atacante con un RST/ACK.

Gráficamente lo vemos como:

Ventajas que presenta:

- Si el servicio está disponible siempre lo encuentra, se podría decir que es infalible.
- No se necesitan permisos de superusuario en el sistema ya que no hay que modificar nada en el segmento que se envía, es el funcionamiento normal.
- Funciona con todos los stacks TCP/IP, por lo que nos sirve para cualquier sistema operativo.

Desventajas que presenta:

- El principal inconveniente que presenta esta técnica es para el atacante, ya que si su intento es positivo siempre queda registrado, dado que es una conexión normal al servicio, como por ejemplo si escaneamos un servidor web, nuestro intento quedará registrado en los logs del servidor de la misma manera en que se registra una petición web normal.

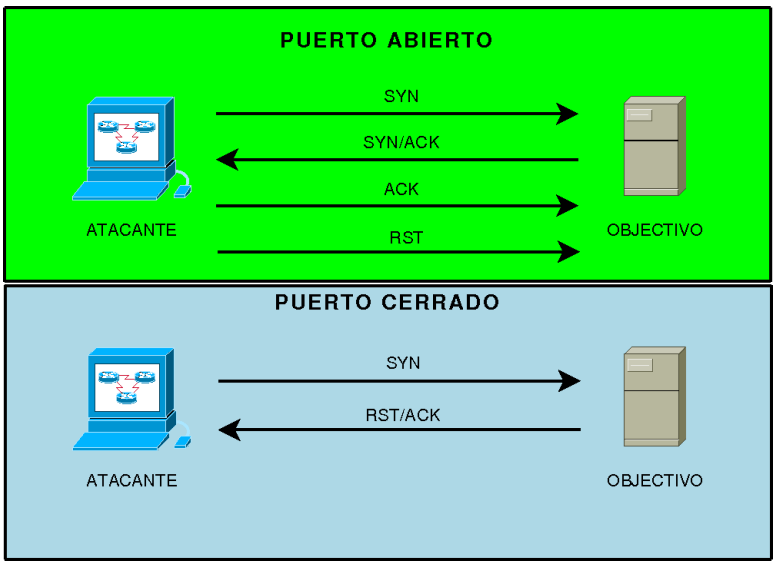

Figura 2.4: Open TCP scanning

Stealth TCP scanning

En esta categoría se agrupan algunas técnicas que se basan en manipular la formación de segmentos, en algunas de ellas malformándolos, enviarlos al objetivo y estudiar las respuestas que se generen, pudiendo identificar de esta manera el estado de los puertos.

La principal diferencia que veremos con el Open TCP scanning es que las técnicas de esta categoría son un poco más difíciles de detectar, ya que procuran pasar despercibidas.

Half-open SYN flag scanning

Esta técnica consiste simplemente en comenzar la apertura de una conexión TCP enviando un SYN, si lo que recibe es una respuesta afirmativa (ACK), inmediatamente aborta la conexión, lo que puede llevarse a cabo o bien enviando un RST o dejando que la conexión se venza por timeout y se considerará el puerto como abierto. Si el puerto está cerrado el objetivo responderá al SYN del atacante con un RST/ACK de la misma manera que en la técnica descripta anteriormente.

Ventajas que presenta:

- Si el servicio está disponible siempre lo encuentra al igual que el método anterior.
- Funciona con todos los stacks TCP/IP, por lo que nos sirve para cualquier sistema operativo.
- Como la conexión no está completamente establecida, la mayoría de las aplicaciones no registrarán la conexión entrante.

Desventajas que presenta:

- Si bien las aplicaciones no loguearán los intentos de conexión, cualquier software de prevención de intrusiones lo detectará fácilmente.

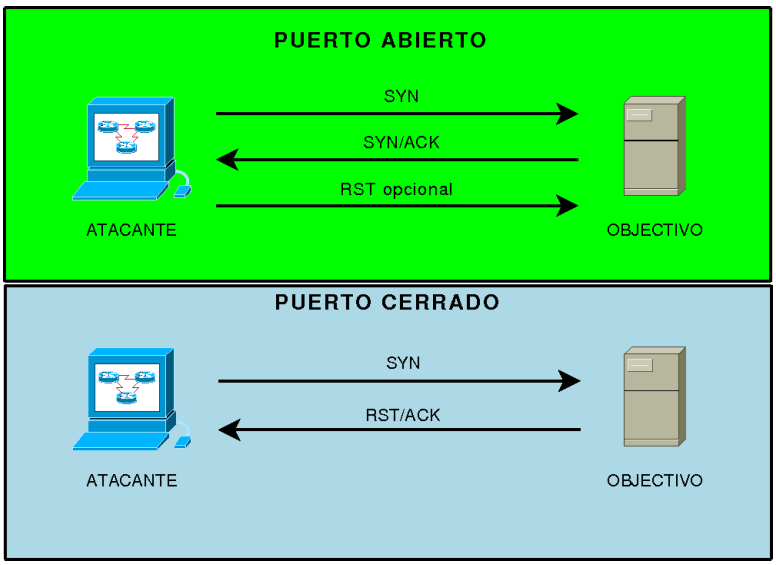

Figura 2.5: Half-open SYN flag scanning

Inverse TCP flag scanning

A estas técnicas que utilizan paquetes con flags de conexiones half-open seteados se las conoce como técnicas inversas porque el host objetivo sólo contestará en caso de que el puerto que estamos probando esté cerrado, si está abierto no recibiremos respuesta. Esto es así porque siguen la RFC 793 que establece que si un puerto está cerrado en un host, cuando éste recibe un requerimiento responderá con un paquete RST/ACK para resetear la conexión.

Resumiendo, consisten en enviar un paquete TCP con un flag inválido a cada puerto que queremos escanear en el objetivo. Normalmente para este tipo de pruebas se utilizan tres configuraciones distintas:

- Una prueba FIN, que es con el flag TCP FIN seteado.
- Una prueba XMAS o árbol de navidad, que es con todos los flags TCP seteados, el FIN, el URG, y el PUSH.
- Una prueba NULL que es sin flags seteados.

Ventajas que presenta:

- Es un método que muchas veces se puede calificar como menos visible, ya que suele ser ignorado por IDS y firewalls, y sobre todo, es invisible para los sistemas de logs de la mayoría de las aplicaciones.

Desventajas que presenta:

- No siempre funciona, muchas veces no detecta correctamente el estado del puerto. Por ejemplo no sirve para los productos de la familia Microsoft, ya que estos sistemas ignoran la RFC 793, y al hacer caso omiso del estándar no enviarán las respuestas RST/ACK ante la consulta a un puerto cerrado.
- Se necesitan permisos de superusuario para poder realizar este tipo de pruebas, ya que es necesario manipular los flags de los paquetes alterando el funcionamiento normal del sistema.

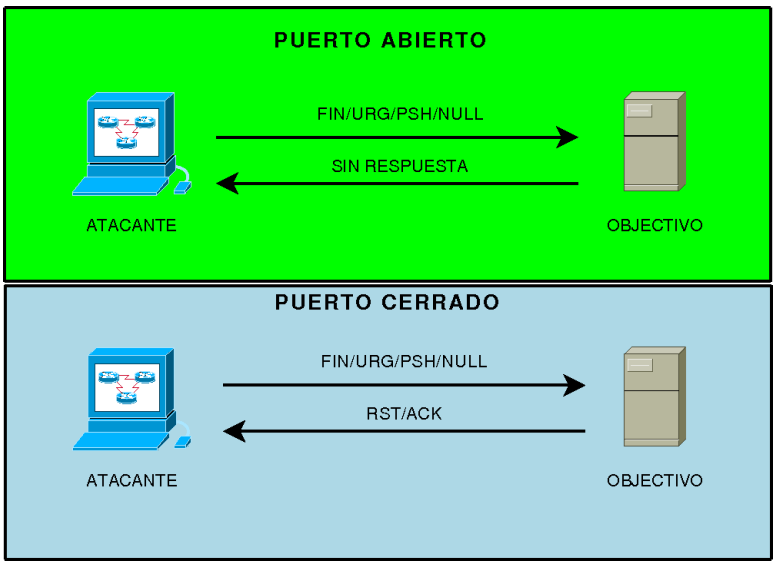

Figura 2.6: Inverse TCP flag scanning

ACK flag probe scanning

Estas técnicas funcionan enviando paquetes de prueba ACK a puertos TCP y analizando los headers de las respuestas RST del objetivo. Mediante este análisis podemos determinar si el puerto está o no abierto. El análisis se centra principalmente en:

- El análisis del campo time-to-live (TTL) de los paquetes recibidos
- El análisis del campo Window de los paquetes recibidos

Para cualquiera de las dos técnicas a utilizar, como se ilustra en la figura 2.7. gráfico, se envían miles de paquetes con flag ACK a distintos puertos del objetivo y se reciben miles de respuestas RST del mismo.

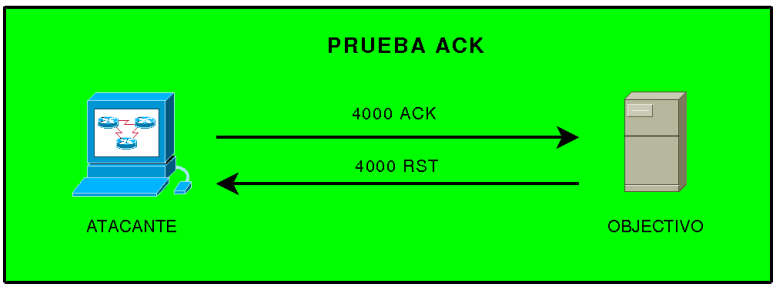

Figura 2.7: ACK flag probe scanning

El análisis del campo time-to-live (TTL) de los paquetes recibidos

Tomamos la muestra de miles de paquetes de respuesta y comparando el campo TTL del header obtendremos qué puertos están abiertos y cuales no.

Por ejemplo, si éstos fueran 4 paquetes de respuesta sucesivos:

- **Paquete 1:** host objetivo port 78: F:RST ->**ttl: 70** win: 0
- **Paquete 2:** host objetivo port 79: F:RST ->**ttl: 70** win: 0

- **Paquete 3:** host objetivo port 80: F:RST ->**ttl: 40** win: 0
- **Paquete 4:** host objetivo port 81: F:RST ->**ttl: 70** win: 0

Podemos observar en el **Paquete 3** que es la respuesta del paquete enviado al puerto **80** tiene un **ttl 40** que es diferente y menor que el resto, lo que nos suele indicar que, al estar por debajo de 64, el puerto estaría abierto.

El análisis del campo Window de los paquetes recibidos

En una muestra similar de miles de paquetes como en el análisis de TTL, pero esta vez comparando el campo windows del header de las respuestas, obtendremos qué puertos están abiertos y cuales no.

Por ejemplo, si éstos fueran 4 paquetes de respuesta sucesivos:

- **Paquete 1:** host objetivo port 78: F:RST ->ttl: 64 **win: 0**
- **Paquete 2:** host objetivo port 79: F:RST ->ttl: 64 **win: 0**
- **Paquete 3:** host objetivo port 80: F:RST ->ttl: 64 **win: 512**
- **Paquete 4:** host objetivo port 81: F:RST ->ttl: 64 **win: 0**

Podemos notar que en todos los paquetes de la muestra el ttl es 64, por lo que podemos concluir que el host objetivo no es vulnerable a este análisis, es decir, observando el TTL del header no podemos decir nada sobre el estado del puerto. En cambio, podemos ver que en el **Paquete 3**, que es la respuesta del paquete enviado al puerto **80** tiene un **win: 512** que es diferente y mayor que el resto, lo que nos suele indicar al ser mayor que 0 que el puerto estaría abierto.

Ventajas que presenta:

- Es un método que muchas veces se puede calificar como invisible, ya que suele ser ignorado por IDS y firewalls, y por sobre todo, es invisible para los sistemas de logs de la mayoría de las aplicaciones.
- Adicionalmente esta técnica nos sirve para chequear si un puerto está filtrado, y puede servirnos también para comprender el esquema de una red compleja, por ejemplo analizando el TTL sabremos cuantos saltos atravesó el paquete hasta llegar desde el objetivo al atacante.

Desventajas que presenta:

- No siempre funciona, muchas veces no detecta correctamente el estado del puerto. En realidad, es utilizable sólo contra sistemas cuyo stack TCP/IP esté basado en BSD, ya que la técnica aprovecha una vulnerabilidad de éste.
- Se necesitan permisos de superusuario para poder realizar este tipo de pruebas, ya que es necesario manipular los flags de los paquetes alterando el funcionamiento normal del sistema.

Third-party and spoofed TCP scanning

Las técnicas que describiremos a continuación presentan dos virtudes desde el punto de vista del atacante que las diferencian de las mencionadas anteriormente: brindan mayor anonimato y posibilitan la evasión de algunas medidas de seguridad. Por un lado las técnicas descriptas en esta sección brindarán mayor anonimato al que está realizando el escaneo ya que todas ellas realizan los análisis a nombre de un tercero. Los escaneos llegarán siempre a los sistemas objetivos como si fueran realizados por terceros y después, de alguna manera, el analizador obtendrá los resultados sin interactuar directamente con el sistema objetivo.

Por otro lado brindan la posibilidad de evadir medidas de seguridad, ya que, como se está usando a otro como origen, tendremos los mismos privilegios que tiene el origen spoofeado y no los del analizador real. Por ejemplo, si el sistema que se está spoofeando está dentro de la LAN del objetivo, las posibilidades de que el escaneo sea filtrado por un firewall son mucho menores.

TCP Idle Scanning (IP ID header scanning)

Esta técnica de escaneo es una de las mas interesantes para llevar a la práctica, se realiza aprovechando algunas peculiaridades de las implementaciones del stack TCP/IP de varios sistemas operativos. Para llevarla a cabo se involucran tres hosts:

- El host que se usa para lanzar el scan, el **ATACANTE**.
- El host que se va a escanear, el **OBJETIVO**.
- Un tercer host, que denominaremos **ZOMBIE**, cuya dirección IP se va a utilizar como la dirección origen del escaneo de puertos. Para la elección del **ZOMBIE** es esencial que ese host esté disponible desde Internet y que tenga poco tráfico. Además de las condiciones necesarias cuanto más cerca esté del host a escanear mejor será, el caso ideal sería que estuviera en la misma LAN que el objetivo.

Funcionamiento:

Este método utiliza un TCP/CONNECT pero de una forma particular. Lo primero que hacemos al elegir nuestro host ZOMBIE es enviarle un paquete de prueba para averiguar el IPID como muestra la figura 2.8

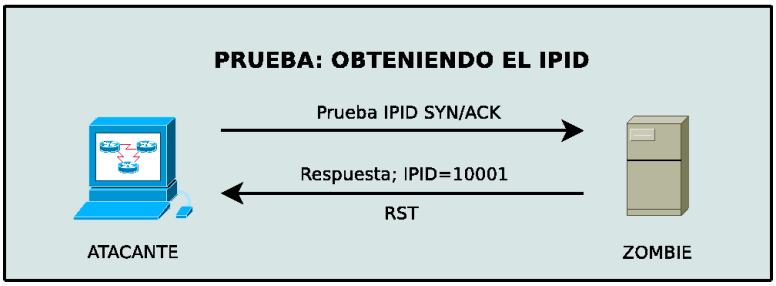

Figura 2.8: TCP Idle Scanning (Zombie obteniendo IPID)

En este ejemplo el ZOMBIE nos devuelve el IPID=10001, valor que guardaremos. Lo siguiente que haremos será enviar un inicio de conexión (SYN) al puerto que queremos saber si está abierto en el host objetivo, pero con el origen del paquete alterado, usando como IP origen la IP del ZOMBIE.

En el caso del puerto abierto el objetivo devolverá un SYN/ACK al origen, que en este caso será el ZOMBIE, ya que fue la IP usada como origen. Cuando el ZOMBIE reciba esta confirmación inmediatamente enviará un RST al objetivo, ya que él no estaba esperando dicho paquete. En este punto hay que prestar particular atención a que cuando se envía el RST se incrementa el IPID. Ver figura 2.9.

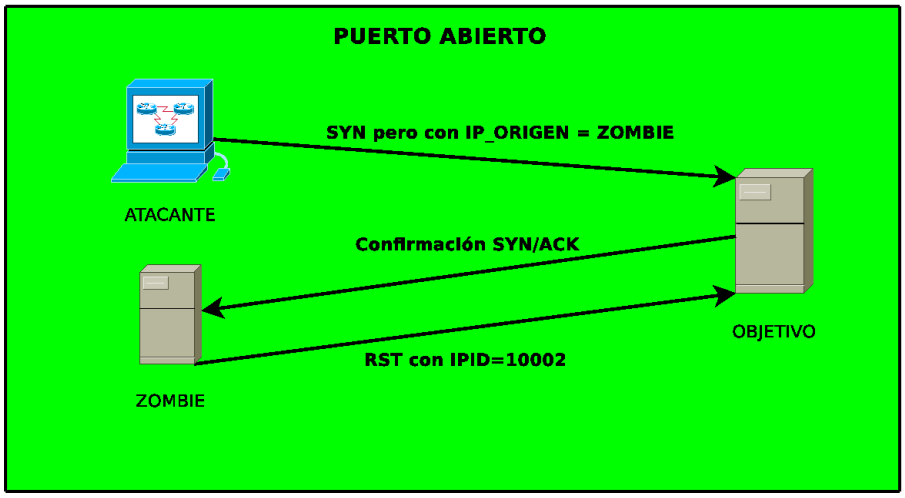

Figura 2.9: TCP Idle Scanning (Zombie para detectar puerto abierto)

Si bien el puerto está abierto, la respuesta del saludo de tres vías (SYN/ACK) llega al ZOMBIE y no al atacante, con lo que el atacante sigue sin saber nada sobre el puerto. Aquí es donde entra a cobrar importancia el IPID, lo que haremos será consultar al ZOMBIE su IPID. Ver figura 2.10.

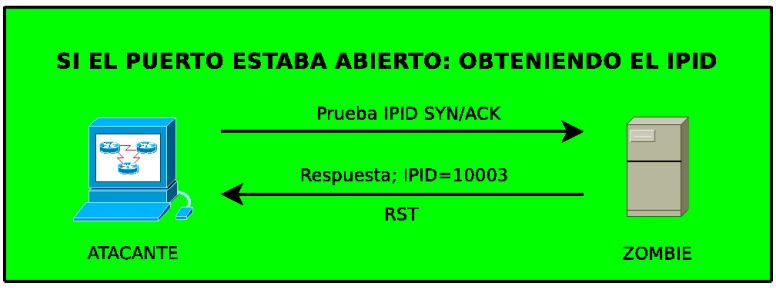

Figura 2.10: TCP Idle Scanning (Zombie obteniendo IPID en puerto abierto)

En el caso del puerto abierto cuando consultamos nuevamente el IPID del ZOMBIE notaremos que se incrementó en 2, lo quiere decir que el ZOMBIE tuvo tráfico extra, con lo que podemos concluir que el puerto estaba abierto. Concretamente se incrementa cuando envía el RST como se puede apreciar en la figura 2.10, recordemos que el IPID de la primer prueba era 10001 y en esta nueva prueba es 10003.

En el caso de que puerto que estamos probando esté cerrado el procedimiento será el mismo, hacemos la prueba del IPID como antes, obtenemos el 10001 original, guardamos este valor y enviamos un inicio de conexión (SYN) al host objetivo al puerto que queremos saber si está abierto, pero con el origen del paquete alterado, usaremos como IP origen la IP del ZOMBIE. Ver figura 2.11.

Como el puerto estaba cerrado, al recibir el SYN el objetivo contesta con un RST al ZOMBIE, ya que ésta es la IP ORIGEN del SYN. Este RST no genera respuesta alguna en el ZOMBIE, por lo que el IPID no se altera. Ver figura 2.12.

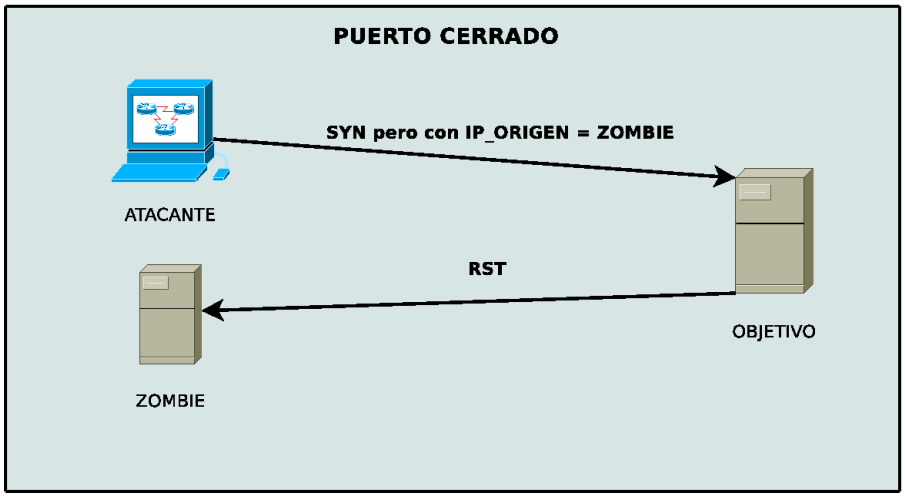

Figura 2.11: TCP Idle Scanning (Zombie para detectar puerto cerrado)

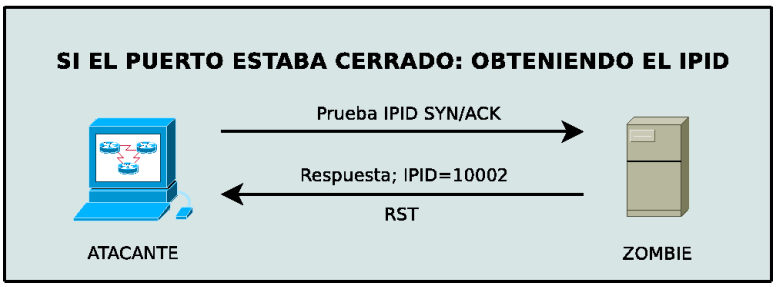

Figura 2.12: TCP Idle Scanning (Zombie obteniendo IPID en puerto cerrado)

Realizando un nuevo test del IPID descubriremos que el valor es el 10002, es decir uno más que en la prueba inicial, lo que indica que el ZOMBIE no tuvo tráfico, por lo que podemos deducir que el puerto está cerrado.

Ventajas:

- Es virtualmente invisible, ya que para el objetivo el atacante nunca se conectó.
- Es muy útil para superar barreras de seguridad como puede ser un firewall, ya que como estamos realizando el ataque desde una máquina ZOMBIE, vamos a abusar de la confianza que tienen en él, tanto el host objetivo como los firewalls que haya que atravesar entre el ZOMBIE y el destino final.
- Si bien suele pensarse que es difícil encontrar una máquina ZOMBIE, que tenga tráfico casi nulo, ésto no es tan cierto, ya que cualquier estación de trabajo de usuario tendrá poco o nada de tráfico en horario no laboral si es que queda prendida. Y, en general estos equipos no tienen sistemas de logs adecuadamente configurados.

Desventajas:

- No funciona en todas las implementaciones de stacks TCP/IP.
- Si bien lograremos un listado de los puertos abiertos, es más difícil conseguir alguna otra información ya que no podemos conectarnos realmente a los servicios.
- Hay que encontrar un ZOMBIE.

Proxy bounce scanning

Este tipo de escaneo se realiza simplemente utilizando un servidor proxy abierto como intermediario de nuestro análisis. Dependerá de la configuración del proxy que tipo de escaneo podemos hacer.

Muchas veces cuando un atacante llega a vulnerar una aplicación web utiliza ese server como proxy para realizar escaneos.

Sniffer-based spoofed scanning

Esta metodología consiste en realizar el escaneo spoofeando la dirección origen con una dirección origen del mismo segmento de red, de esta manera lo que vamos a tener que hacer para saber el resultado de los intentos es sniffear la red esperando por las respuestas de los host objetivos. De esta manera podemos utilizar casi todas las técnicas antes descriptas.

Ventajas:

- Podemos decir que es invisible para los logs, cuanto más grande sea el segmento de red en el que estamos más difícil será ubicarnos ya que podemos usar por ejemplo una IP de otra oficina de una PC a la que no tenemos acceso para evitar sospechas o incluso un dirección no utilizada.
- Podemos abusarnos de la confianza entre las máquinas para conseguir resultados, por ejemplo, si hacemos el escaneo con IP origen la máquina del administrador de la red vamos a tener acceso a todo lo que esté habilitado para esa IP.

Desventajas

- Se deben tener permisos de administrador en la máquina desde donde se quiera lanzar el ataque, ya que habrá que poner la placa de red en modo promiscuo para poder sniffear la red.
- Corremos el riesgo de que al poner la placa en modo promiscuo nos detecte algún IDS.
- Con el tiempo las redes migraron de redes de hubs en la que no había separación a nivel de capa de enlace a redes switcheadas, lo que hizo un poco mas difícil este ataque, aunque no imposible.

Técnicas de scanning para puertos UDP

Vimos que para puertos TCP hay diversas técnicas para realizar análisis de estado, por el contrario en UDP como es un protocolo sin conexión solo tenemos dos maneras efectivas para saber si un puerto esta abierto o cerrado.

Buscando respuestas negativas de ICMP[17] "destination port unreachable"

Simplemente se envían requerimientos al subconjunto de los 65535 puertos UDP que se quieren escanear y se espera respuestas. En el caso de puertos abiertos no se va recibir respuesta alguna y en el caso de los puertos cerrados recibiremos un mensaje **ICMP "destination port unreachable"**. Ver figura 2.13.

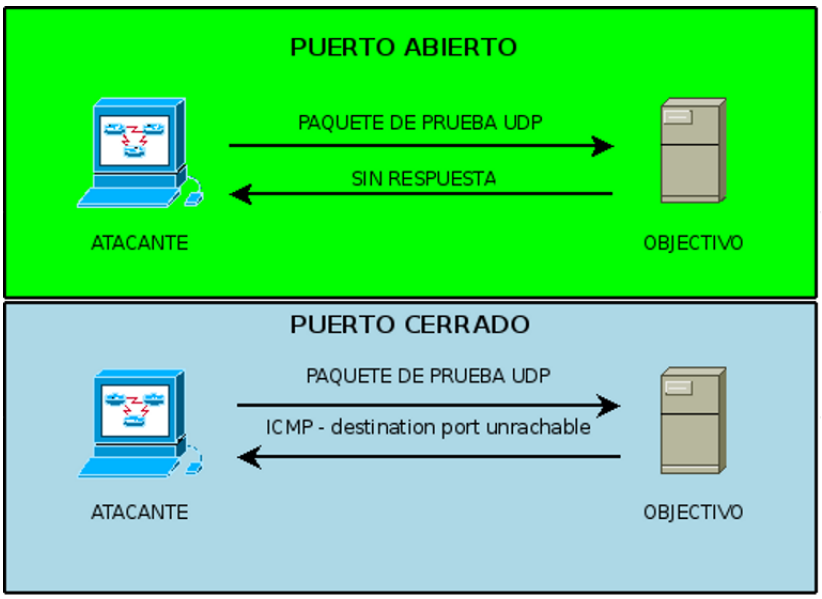

Figura 2.13: UDP Scanning

La principal problemática de esta técnica es que muchas organizaciones filtran los mensajes ICMP hacia afuera de la red local, o incluso las mismas máquinas de usuarios conectadas a Internet directamente con conexiones hogareñas también lo hacen en el firewall, lo cual ocasiona muchos falsos positivos. Podemos decir que esta es una técnica de scanning inverso.

Generando positivamente la conexión, para lo que se necesita un cliente del protocolo

Simplemente utilizamos un cliente, por ejemplo de TFTP[18] e intentamos conectarnos, si el intento es exitoso el puerto esta abierto y el servicio funcionando, y si no lo logramos entonces está cerrado.

Escaneador de sistemas operativos

La idea es obtener el Sistema Operativo y la versión específica que está corriendo en el host objetivo. Hay distintas técnicas utilizadas para obtener esta información y, en general, los mismos escaneadores de puertos vistos más arriba incluyen esta funcionalidad.

Fingerprints

Un OS fingerprint es considerado como la huella digital de un sistema operativo. Consiste en estudiar el comportamiento del stack TCP/IP de una máquina y en base a la manera en que reacciona ante distintos eventos podemos determinar que SO está corriendo en ella. Esta diferenciación es posible ya que casi todos los stacks se implementan de manera distinta, principalmente por dos motivos: primero porque muchos de los RFC tienen puntos no definidos con total exactitud, entonces quedan a criterio de los programadores de los sistemas operativos a la hora de la implementación, y en segundo lugar porque muchas implementaciones de SO directamente no siguen los RFC.

Active Fingerprint

Lo que se hace habitualmente es generar tráfico al host que se está analizando y estudiar su comportamiento. Las aplicaciones que implementan active fingerprint funcionan en base a la información de puertos abiertos y cerrados recogida, generando tráfico a distintos puertos TCP y UDP, y algún requerimiento ICMP y analizando las respuestas. Hay muchísimas técnicas de generación de tráfico para testear las distintas combinaciones de respuestas que pueden generarse en el stack de cada SO sobre las cuales vamos a evitar profundizar en este trabajo. En base a las respuestas recibidas se realiza una combinación y se obtiene lo que se llama un OS Fingerprint, información que luego es comparada con una base de datos de la herramienta hasta poder corresponderlo con una entrada existente y dar como respuesta el sistema operativo y la versión específica que corre en el host.

Esta es una de las técnicas más ampliamente aceptadas ya que nos brindará la suficiente seguridad en la respuesta y con una velocidad aceptable, aunque tiene la contra de que el atacante o analizador queda expuesto abiertamente ante el objetivo.

Passive Fingerprint

El objetivo de esta técnica es formar un OS fingerprint similar al de la técnica anterior. Se pueden analizar los mismos parámetros en los paquetes, con la diferencia que no se genera tráfico para recibir las respuestas, sino que lo que se hace es escuchar la red en busca de tráfico TCP, UDP e ICMP que nos permita hacer la combinación que genera el OS Fingerprint. Una vez obtenido, de igual manera que en el método activo se consulta la base de datos en busca del SO correspondiente.

La ventaja esencial de este método es que, al no generar tráfico hacia el objetivo directamente, se evita quedar en evidencia.

Entre las desventajas tenemos que posiblemente este método sea más lento que el anterior, y que además será necesario que estén dadas las condiciones para poder sniffear el tráfico de red del objetivo.

Otras técnicas

Hay muchas otras técnicas que en general se usan menos que las anteriores, algunos ejemplos son:

Identificación en base a los banners

Esta técnica es bastante antigua, y hoy podríamos calificarla hasta de ingenua, consiste básicamente en leer los banners que presentan las distintas aplicaciones que se ejecutan en un host, y en base a ello identificar el sistema operativo que está corriendo. Un banner es la presentación de una aplicación, por ejemplo que cuando nos atiende el FTP[19] nos dice “Bienvenidos a FTPD versión Y.Y para Linux Debian”. Como los banners son configurables por el administrador y de hecho algunas medidas de seguridad son o bien alterar los banners poniendo cualquier cosa o bien directamente suprimirlos es que decimos que es una técnica ingenua, ya que sus resultados son muy poco confiables.

Open Port Patterns

En el caso de Open Port Pattern, en base a los puertos abiertos, se presupone que es un SO u otro, por ejemplo si está abierto el puerto 22 y el puerto 80 podría

decir que es un Linux, ya que Linux en general tiene OpenSSH[20] en el puerto 22 y Apache[21] en el puerto 80, pero ambas aplicaciones están portadas a Windows, o sea que podríamos tener un Windows con OpenSSH en el puerto 22 y Apache en el puerto 80, lo cual genera confusión y no permite identificar a ciencia cierta el sistema operativo del objetivo.

Exploit chronology

Esta técnica es mas peligrosa ya que básicamente consiste en generar DoS (Denial of Service). Dado que no se puede distinguir entre el TCP stack de Windows 95, 98 y NT, porque no lo han actualizado en lo más mínimo, o sea el NT sigue teniendo el mismo stack que el 95 y que el 98 es que surgió esta técnica. Consiste en hacer un ataque de DoS, en general se empieza con los más antiguos (un Nuke o un Ping de la Muerte) y luego se usan técnicas un poco más avanzadas (como Teardrop[7] o Land[8]). Después de cada ataque se hace un PING a la víctima y se analiza si responde o si se logro la caída del host. Si cayó se analiza cual fue la técnica que lo logró, y luego se compara con el historial de actualizaciones de los Stacks para así deducir qué sistema es e incluso hasta que Service Pack o Hotfix tiene aplicado.

La desventaja principal es que causará el DoS del objetivo.

Detección de versiones de servicios y aplicaciones

De la misma forma que con los mecanismos de detección relatados antes para detectar el SO instalado, hay diversas técnicas para conseguir la versión específica de un servicio que está brindando en la red el equipo objetivo, a continuación nos referiremos a aquellas que son de público conocimiento.

Para obtener la versión del software que se está ejecutando será necesario interactuar con el objetivo, ya que ésta es la única manera de obtener dicha información, así que todo lo invisible que hubiera sido nuestro escaneo de puertos anterior se perderá. Por ejemplo, si escaneamos utilizando la técnica del zombie host y descubrimos que el puerto TCP 80 del objetivo está abierto, ahora para saber que versión está corriendo vamos a tener que conectarnos directamente a él, con lo que por primera vez estaríamos revelándonos ante el objetivo.

Teóricamente la solución a este comportamiento activo y revelador sería o bien algún tipo de conexión spoofeando la dirección origen y sniffeando las respuestas, o bien utilizar algún proxy para conectarnos con el objetivo.

NULL Probe

Si el puerto es TCP, lo primero que hay que hacer es conectarse a él, en cuyo caso se pierde el anonimato.

Una vez que la conexión está establecida, el escaneador espera escuchando por unos pocos segundos. Durante ese lapso muchas implementaciones de servicios tales como FTP, SSH[22], SMTP[23], Telnet[24], POP3[25] o IMAP[26] se identifican a ellos mismos con un mensaje de bienvenida. Este banner en general incluye alguna información como la versión específica del servicio que se está prestando. Esa respuesta es comparada con una base de firmas de servicios, lo que en realidad son

[7] Teardrop: ataque de DoS manipulando la fragmentación IP

[8] Land: ataque de DoS manipulando los puertos e ips de los paquetes

listas de miles de expresiones regulares que se intentarán matchear con la información recibida, y al obtener una coincidencia se sabrá que aplicación y que versión es la que está ejecutándose.

En general, si el servidor envía el banner al atacante, se puede tener dos tipos de resultados positivos y uno negativo. Entre los positivos se puede llegar a una identificación completa o una identificación parcial del servicio. Diremos que el resultado es negativo cuando directamente no se puede identificar el servicio. En el caso que la identificación sea total podemos dar por concluido el test, dado que ya se tiene la informaciión deseada. En el caso de que sea parcial o negativo hay que proseguir con otro tipo de test ya que se necesita mas información. La identificación parcial se conoce como soft-match.

En el caso que nunca se envíe el mensaje de bienvenida el test fracasará.

Se denomina NULL Probe a este tipo de test ya que no se envía ningún dato al servidor, lo único que se hace es abrir la conexión TCP y esperar.

Pruebas activas

Este tipo de pruebas, en general, se llevan a cabo para aplicaciones ejecutándose en puertos UDP y para aplicaciones en puertos TCP para las que las NULL probes no proveyeron una identificación positiva completa, ya sea porque fallaron o porque se obtuvo un soft-match.

Para ahorrar tiempo y procesamiento innecesario, habitualmente se acota el universo de pruebas que se hacen sobre cada puerto. Se tiene una lista de pruebas asociadas a una lista de puertos específica que será donde habitualmente corre la aplicación y cuando se esté analizando un puerto solo se ejecutarán las pruebas asociadas. Por ejemplo, si se está escaneando el puerto TCP 80 se enviarán pruebas sólo para servidores web, que es el tipo de aplicación que habitualmente está en ese puerto. A su vez, si ya se tenía un soft-match, el universo de pruebas se acota aún más, ya que se realizarán en base a la información parcial obtenida. Por ejemplo, si ya sabemos que el servidor web que se está ejecutando es Apache, se ejecutarán las pruebas para saber la versión exacta y no todas las pruebas para las distintas implementaciones de servidores webs.

Cada prueba incluirá una cadena de caractéres de prueba que se envía al puerto a testear, la respuesta que se obtiene se compara con una lista de expresiones regulares de la misma manera que fue descripto para NULL probe. Y, al igual que en el método anterior, podemos obtener como resultado una identificación completa (en cuyo caso el test termina), una identificación parcial (en cuyo caso se redefinen las pruebas orientándolas a la aplicación en particular) o ninguna identificación, ésto dependerá directamente de la base de expresiones regulares que tenga la herramienta de análisis.

Este tipo de prueba sirve como complemento del escaneo de puertos UDP ya que nos servirá para verificar los puertos UDP abiertos que el análisis previo pudo haber catalogado como filtrado debido a una configuración específica de algún firewall intermedio.

Convivencia entre ambas técnicas

Analizando TCP en muchos casos la prueba NULL y la prueba activa pueden compartir la misma conexión, iniciada por la primera, ésto se hace para evitar aparecer múltiples veces en los registros evitando ser tan visible. En el caso de que haya que hacer múltiples pruebas activas, por ejemplo cuando el resultado da soft-match,

puede ocurrir que haya que generar una conexión por cada prueba activa, ya que el resultado de una puede corromper el análisis de otra.

2.3.3. Presentación de resultados

Si bien dependerá exclusivamente de la implementación del analizador de vulnerabilidades que estemos usando, en general los resultados obtenidos al terminar el análisis se presentan con referencia a los índices adoptados mundialmente para los problemas de seguridad.

Para poder identificarlos inequívocamente se utilizan diversos códigos, de acuerdo al ámbito en que se lo quiera mencionar, la mayoría de las veces una misma vulnerabilidad tendrá asociado diversos identificadores.

Índices globales:

- **CVE - Common Vulnerabilities and Exposures[27]**: Son identificadores únicos para referirse a vulnerabilidades de seguridad públicamente reconocidas como tales, es mantenido por el U.S. Department of Homeland Security. Cada identificador CVE incluye:

  - Un número identificador CVE (ejemplo: “CVE-1999-0067”)
  - Indicación del estado actual: si es “entry” o ”candidate”
  - Breve descripción de la vulnerabilidad
  - Referencias que se consideren pertinentes

  En general cuando se descubre una nueva vulnerabilidad se le asigna un nuevo identificador y se setea el estado en “candidate“ que indica que el reporte aún no fue analizado pero que pretende ingresar al listado de CVEs. Esta vulnerabilidad candidata será analizada por parte de la CVE Editorial Board, que es un grupo conformado por personas de diversas reconocidas organizaciones dedicadas a la seguridad. En caso de pasar el estudio se le asignará un status de ”entry“ pasando a formar parte del listado, en caso contrario continuará como “candidate“ y los motivos serán publicados junto con el reporte.

  Anteriormente, antes del 19 de octubre de 2005, los reportes no tenían status, sino que se diferenciaban por el identificador, se utilizaba CAN en lugar de CVE, y se reemplazaba el CAN por el CVE una vez aprobada la vulnerabilidad. Por ejemplo la CVE-2002-0649 antes de ser entrada de la lista de CVE se identificaba como CAN-2002-0649.

  Ejemplo de identificador: El id CVE-2002-0649 indica un Buffer overflow en ASP, en la función Include que afecta a varias versiones de IIS. A través de ese ID podemos fácilmente obtener información pública sobre la vulnerabilidad a través de la página de http://cve.mitre.org.

- **BID - Bugtraq ID**: Son identificadores que permiten ubicar información sobre algún problema de seguridad en la página de la empresa Security Focus[28], en general tienen su correspondencia con algún o algunos identificador de CVE, por ejemplo el CVE citado anteriormente CVE-2002-0649 lo encontramos relacionado con los Bugtraq ids 5310 y 5311. Estos identificadores son asignados directamente por la empresa a partir de la información que circula en sus listas de correo, las cuales son de suscripción pública gratuita y con mucha actividad.

Por cada Bugtraq id tendremos bastante información presentada ordenadamente: información general sobre el bug como por ejemplo el número de CVE con el que se relaciona y el software que afecta, discusiones en la lista de mails de Security Focus que trataron sobre el problema, el exploit para el problema, la solución y referencias.

Índices según el origen del reporte: Dependiendo del origen del reporte tendremos identificadores de distintos orígenes que generalmente mapean con un CVE o un BID, por ejemplo:

- **USN-432-1** es un reporte de seguridad de Ubuntu, el Ubuntu Security Notice con identificador USN-432-1 que se corresponde con el **CVE-2007-1263**
- **MDKSA-2007:055** es un reporte de seguridad de Mandriva, el Mandriva Linux Security Advisory MDKSA-2007:055 que se corresponde con el **CVE-2007-1246**
- **MS07-008** es un reporte de seguridad de Microsoft, publicado en el Microsoft Security Bulletin MS07-008 el cual se corresponde con el **CVE-2007-0214**

Otros: En general todas las distribuciones de GNU/Linux grandes tienen un código identificable propio:

- Referencia Debian: DSA-1168-1
- Referencia Mandriva: MDKSA-2006:155
- Referencia Red Hat: RHSA-2006:0633-5
- Referencia Suse: SUSE-SA:2006:050
- Referencia Ubuntu: USN-340-1
- Referencia Gentoo: GLSA-200609-0
- Referencia Slackware: SSA:2006-217-01

A través de estos códigos podemos encontrar muy fácilmente información sobre una vulnerabilidad en particular, la forma de solucionar el problema y a veces hasta la forma de explotarla e incluso el exploit para hacerlo.

Capítulo 3

Criterios de selección de herramientas

Para la selección de los productos tuvimos en cuenta varios criterios:

- la licencia con la que se distribuye el software
- la comunidad que lo utiliza
- la documentación existente
- las pruebas realizadas cuando quisimos integrarlos
- la facilidad de instalación
- la experiencia previa en el uso de los mismos

3.1. La licencia

La licencia[29] bajo la que se distribuye el software a utilizar es tal vez el único criterio insalvable, ya que si usamos programas cuya licencia nos implique algún costo para su uso o nos genere alguna prohibición de uso y/o lectura y/o modificación del código fuente estaríamos ante un escollo importante.

Si la traba sólo fuera de costo sería tal vez un problema eludible, ya que pagando la licencia el problema se soluciona, pero sigue sin ser un ideal que perseguimos con la implementación, que el costo de licenciamiento sea de cero pesos.

Ahora si la traba es con algún impedimento de uso, lectura o modificación si tenemos un problema más importante. Si es de uso, por ejemplo que no esté permitido usar la salida del producto en otro sistema se nos complicaría mucho la integración, pero si la restricción es de lectura o modificación de los fuentes estaríamos aún peor. El escenario ideal para nuestro trabajo es que la licencia que tienen los productos respeten la mayor cantidad de puntos que establece la Fundación del software libre[30] como las libertades básicas.

La Definición de Software Libre

El software libre[31] es la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Más precisamente, se refiere a cuatro tipos de libertades para los usuarios del software:

1. La libertad de ejecutar el programa, para cualquier propósito (libertad 0).

Figura 3.1: El logo de GNU

2. La libertad de estudiar cómo trabaja el programa, y adaptarlo a sus necesidades (libertad 1).

3. La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).

4. La libertad de mejorar el programa y publicar sus mejoras, y versiones modificadas en general, para que se beneficie toda la comunidad (libertad 3).

Para que las libertades para realizar cambios, y publicar versiones mejoradas, tengan sentido, se debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el software libre.

Un programa es software libre si los usuarios tienen todas estas libertades. Entonces, debería ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratis o cobrando una tarifa por distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

También debería tener la libertad de hacer modificaciones y usarlas en privado, en su propio trabajo u obra, sin siquiera mencionar que existen. Si publica sus cambios, no debería estar obligado a notificarlo a alguien en particular, o de alguna forma en particular. Para que estas libertades puedan ser reales, deben ser irrevocables; si el programador del software tiene el poder de revocar la licencia, o de cambiar retroactivamente sus términos, el software no es libre.

¿Qué es el copyleft?

Ciertos tipos de reglas sobre la manera de distribuir software libre son aceptables, cuando no entran en conflicto con las libertades principales. Por ejemplo, el copyleft[32] es la regla en base a la cual, cuando se redistribuye un programa, no se pueden agregar restricciones para denegar a las demás personas las libertades principales. Esta regla no entra en conflicto con las libertades principales; más bien las protege. Copyleft es la forma general de hacer un programa software libre y requiere que todas las modificaciones y versiones extendidas del programa sean también software libre.

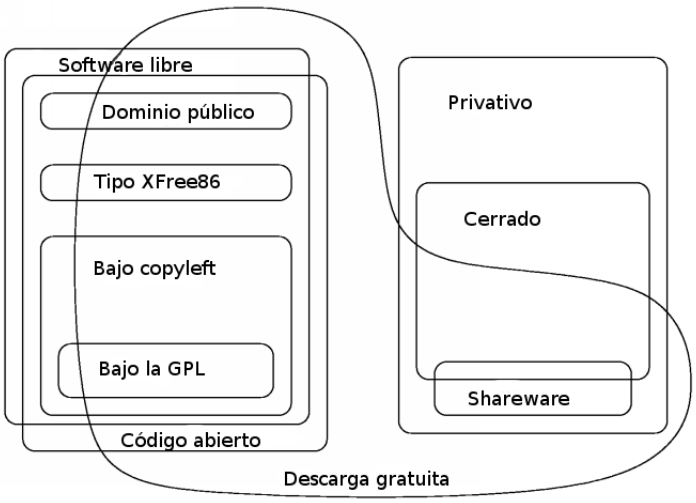

Figura 3.2: Categorías del software

Categorías de software libre y no libre[33]

1. **Software libre:** software que cumple las 4 libertades para los usuarios antes descriptas.

2. **Software de código abierto (Open source)[34]:** El software de código abierto no significa solo acceso al código fuente. Los términos de distribución deben cumplir varios criterios:

   a) Debe permitirse la redistribución libre.

   b) El software debe incluir el código fuente.

   c) Se deben permitir trabajos derivados.

   d) Mantener la integridad del código fuente del autor, los trabajos derivados deberán llevar otro nombre u otro numero de versión que el software original.

   e) No se permite la discriminación contra personas o grupos.

   f) No se debe restringir su uso en un campo de acción en particular.

   g) Los derechos asociados al programa deben aplicarse a todos aquellos a quienes se redistribuya el programa, sin necesidad de pedir una licencia adicional.

   h) La licencia no debe ser específica de un producto.

   i) La licencia no debe restringir otro software.

   j) La licencia debe ser neutral tecnológicamente hablando.

   Mucha gente utiliza la expresión software de "código abierto" para referirse, más o menos, a la misma categoría a la que pertenece el software libre. Sin embargo, no son exactamente el mismo tipo de software: el software de código abierto acepta algunas licencias que el software libre considera demasiado restrictivas, y hay licencias de software libre que software de código abierto no ha aceptado.

3. **Software de dominio público:** El software de dominio público es software que no está protegido por derechos de autor. Es un caso especial de software libre no protegido con copyleft, lo que significa que algunas copias o versiones modificadas pueden no ser completamente libres.

4. **Software protegido con copyleft:** El software protegido con copyleft es software libre cuyos términos de distribución aseguran que todas las copias de todas las versiones son software libre.

5. **Software libre no protegido con copyleft:** El software libre no protegido con copyleft, incluye la autorización del autor para redistribuir y modificar el software, así como el permiso para añadirle restricciones adicionales. Que un programa sea libre pero no esté protegido con copyleft, implica que algunas copias o versiones modificadas del mismo pueden no ser completamente libres. Una compañía de software podría compilar el programa, con o sin modificaciones, y distribuir el archivo ejecutable como un producto de software privativo.

6. **Software cubierto por la GPL:** La GPL (General Public License/Licencia Pública General)[35] de GNU[3] es un conjunto específico de términos de distribución empleados para proteger un programa con copyleft. El Proyecto GNU utiliza esta licencia para la distribución de la mayoría del software de GNU.

7. **El sistema GNU[36]:** El sistema GNU es el sistema operativo similar a Unix, constituido en su totalidad por software libre, que se ha desarrollado en el Proyecto GNU desde 1984. Éste incluía el Hurd de GNU como núcleo, desarrollado desde 1990. El sistema GNU/Linux[37], es un derivado del sistema GNU que utiliza Linux como núcleo en vez del Hurd de GNU.

   Ya que el propósito de GNU es ser un sistema libre, cada una de las piezas que lo componen deben ser software libre. Sin embargo, no todas tienen por que estar protegidas por copyleft.

8. **Programa o Software de GNU:** El software de GNU es el software liberado bajo el auspicio del Proyecto GNU.

9. **Software no libre:** El software no libre es cualquier software que no es libre. Esto incluye al software semilibre y el software privativo.

10. **Software semilibre:** El software semilibre es software que no es libre, pero incluye autorización para que los particulares lo usen, lo copien, lo distribuyan y lo modifiquen (incluyendo la distribución de versiones modificadas) sin propósitos lucrativos. Las restricciones del copyleft están diseñadas para proteger las libertades esenciales de todos los usuarios. La única restricción substantiva justificada en el uso de un programa es la que previene la adición de restricciones por parte de otras personas. Los programas semilibres tienen restricciones adicionales motivadas por fines puramente egoístas.

11. **Software privativo:** El software privativo es software que no es libre ni semilibre. Su uso, redistribución o modificación están prohibidos.

12. **Freeware:** El término "freeware" es usado comúnmente para referirse a paquetes que se pueden distribuir pero no modificar (y cuyo código fuente no está disponible).

13. **Shareware:** El Shareware es software del que se permite redistribuir copias, pero que por cada copia utilizada, el usuario debe pagar un cargo por licencia.

14. **Software privado:** El software privado, o a medida, es software desarrollado para un usuario (generalmente una organización o una compañía). Este usuario lo tiene en su poder y lo utiliza, y no lo libera al público ni como código fuente ni como binario.

15. **Software comercial:** El software comercial es aquel desarrollado por un negocio que pretende obtener dinero de su utilización. ¡"Comercial" y "privativo" no son lo mismo! La mayoría del software comercial es privativo, pero hay software libre comercial, y hay software no libre no comercial.

Más allá del software

Los manuales de software deben ser libres, por las mismas razones que el software debe ser libre, porque en efecto los manuales son parte del software. La mayor deficiencia de los sistemas operativos libres no es el software, es la falta de buenos manuales libres que se puedan incluir en estos sistemas. Muchos de los programas más importantes no vienen con manuales completos. La documentación es una parte esencial de cualquier paquete de software; cuando un paquete importante de software libre no viene con un manual libre, es un vacío importante.

Las licencias libres

Para que el software que se publica sea software libre tiene que ser publicado con una licencia de software libre.

- **Licencia Pública General de GNU (GPL):** La Licencia Pública General de GNU, llamada comúnmente GNU GPL, es de tipo copyleft, la usan la mayoría de los programas de GNU y más de la mitad de las aplicaciones de software libre.

- **Licencia Pública General Reducida de GNU (LGPL):** Es una licencia de software libre, pero no tiene un copyleft fuerte, porque permite que el software se enlace con módulos no libres.

- **Licencia de Documentación Libre de GNU (GFDL):** Esta es una licencia pensada para el uso de documentación libre con copyleft.

3.2. Experiencia previa propia

Al momento de elegir tanto las herramientas que integramos como las herramientas que usamos para el desarrollo le dimos un peso significativo a la experiencia propia. Tratamos de utilizar los ambientes de trabajo, lenguajes de desarrollo y aplicaciones con los que nos sentimos más cómodos y tenemos más experiencia, ya que el uso y la resolución de problemas se hace un poco menos complejo.

Para comprender de que hablamos cuando citamos experiencia propia hacemos un pequeño resumen de nuestra historia con el software libre.

Ambos venimos del mundo privativo y comercial del software, nuestra educación universitaria se basó en un altísimo porcentaje en la utilización de productos

Microsoft como base (nos animaríamos a decir que un 98 %). Gracias a nuestras primeras experiencias laborales nos tuvimos que introducir en el mundo de GNU/Linux, primero en su uso en servidores y de a poquito con el paso del tiempo el mundo del software libre fue reemplazando todo lo privativo que usábamos tanto en el trabajo como en nuestros hogares, hasta reemplazarlo hoy casi en su totalidad, incluso en nuestros celulares hemos realizado algunas pruebas con Android[38].

En el camino más allá de ser usuarios de una gran variedad de productos de libres todos los días, nos hemos vuelto desarrolladores del software libre, participando de proyectos como Lihuen GNU/Linux[39] y KOHA-UNLP[40]. También nos hemos vuelto un poco predicadores, asistimos a charlas en colegios y en conferencias, tratamos de convencer a nuestros amigos, familiares y compañeros de trabajo, y en las cátedras en que participamos y tenemos injerencia incentivamos el uso de esta categoría de software todo lo posible.

Por todo lo nombrado podemos decir que la experiencia es un factor significativo en la elección de lo que vamos a usar.

3.3. El idioma

En este desarrollo el idioma en que están las aplicaciones y la documentación de las mismas no es un factor con un peso significativo, ya que si bien este ítem en general es una traba para el usuario final del sistema, los usuarios del nuestro no se verán afectados, debido a que el desarrollo que planteamos presenta una capa de abstracción entre las herramientas que se integran y lo que ve el usuario cliente.

3.4. La comunidad que lo soporta

Tanto la comunidad de desarrolladores como la comunidad de usuarios que usan cada uno de los productos es muy importante para el proceso de selección.

La comunidad de desarrolladores es muy importante porque son el indicador del soporte y la actualidad que tiene un sistema, cuanto más desarrolladores haya detrás más fácil será conseguir la solución a los problemas que aparezcan y es más difícil que el desarrollo se estanque o "muera" al quedarse sin gente detrás que lo continúe. ¿De qué me sirve un analizador de vulnerabilidades si nadie desarrolla nuevos plugins para detectar vulnerabilidades?

La comunidad de usuarios que hay detrás del soft es también muy importante por varios factores. Podemos citar por ejemplo que representan un indicador de la aceptación del producto, o que al haber más usuarios se genera más documentación ya que hay más consultas y por lo tanto más respuestas, o que muchas veces en la comunidad del software libre los mismos usuarios de los productos se convierten de alguna manera en parte del equipo de desarrollo al aportar ideas, documentación, testing, problemas y/o soluciones.

3.5. La facilidad de instalación

Como la idea es que una vez terminado el desarrollo, se pueda hacer puestas en producción con cierta facilidad tomamos como parámetros de importancia que tan difícil es poder instalar y configurar los distintos productos a integrar. Esto hubo que tenerlo en cuenta tanto en la instalación de un producto por separado como para

evaluar su convivencia con el resto del sistema, ya que suele encontrarse software que aislado funcionan sin mayores problemas pero que cuando tiene que convivir con otras aplicaciones interfiere drásticamente con ellas.

Capítulo 4

Productos elegidos para integrar

4.1. El sistema operativo donde se alojará la aplicación

4.1.1. El filtro inicial

En principio, respetando el precepto de las licencias libres, nuestras opciones se reducen considerablemente, ya que quedan descartado todo sistema operativo que sea distribuido bajo una licencia **no libre**. Por esto cualquier sistema de la empresa Microsoft y varios sistemas de la rama *NIX como por ejemplo RedHat o cualquier UNIX quedan descartados inmediatamente.

Entonces, llegado este punto, utilizando como primer criterio de selección la licencia de distribución, nuestras opciones se fueron reduciendo a algún sistema GNU/Linux o alguno de la rama derivada de BSD[41].

4.1.2. Basados en BSD

El Berkeley Software Distribution (BSD, conocido también como el Berkeley Unix)[42][43] es un sistema operativo derivado de UNIX desarrollado y distribuido entre los años 1977 y 1995 por el Computer Systems Research Group (CSRG) de la Universidad de California, Berkeley.

Históricamente se lo ha considerado una rama de UNIX, el UNIX BSD, debido a que inicialmente compartía con el UNIX original de AT&T el código y el diseño, a partir de lo que fue desarrollándose hasta transformarse en un reemplazo libre.

Si bien BSD ha realizado varias contribuciones al desarrollo en el campo de los sistemas operativos, entre las más importantes podemos nombrar:

- El manejo de memoria virtual paginado por demanda
- El control de trabajos
- El Fast FileSystem
- El protocolo TCP/IP (casi todas las implementaciones de TCP derivan de la de 4.4BSD-Lite)

A través de los años BSD fue pasando de versión en versión desde la primer versión 3BSD publicada en 1979 hasta la 4.4BSD-Lite release 2 que fue la última realizada con el CSRG en el año 1995. Ver imagen 4.1.

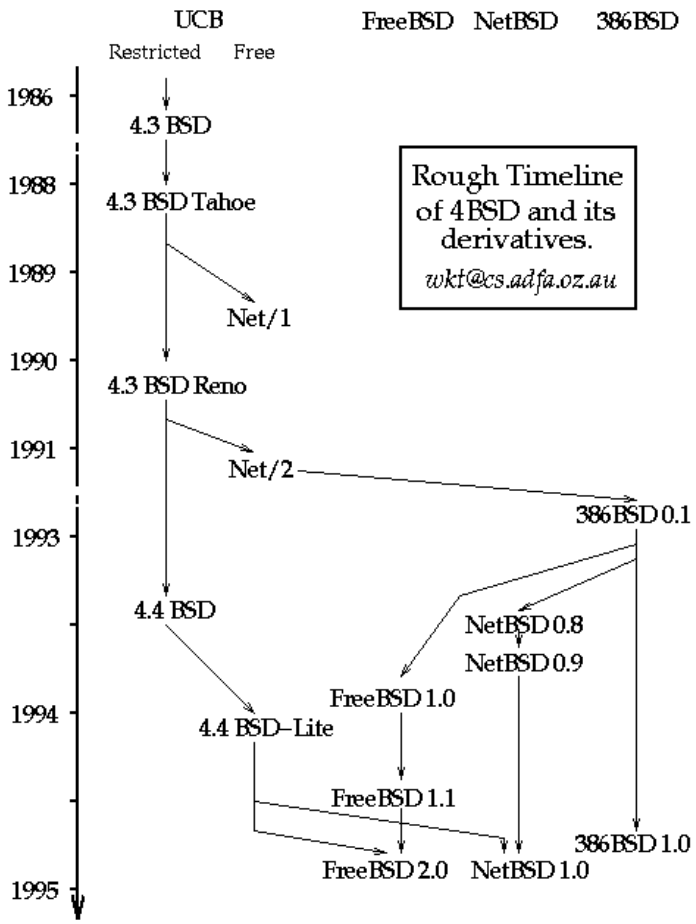

Figura 4.1: Evolución de BSD

Al quedar discontinuado el desarrollo de Berkley, actualmente se utiliza el término BSD para referirse a alguno de sus famosos descendientes[44], algunos distribuidos bajo licencia libre y otros bajo licencia comercial.

Entre los descendientes más famosos podemos nombrar:

- Apple Mac OS X
- The DragonFly BSD Project
- FreeBSD
- m0n0wall
- The NetBSD Project
- The OpenBSD Project
- OpenDarwin
- PC-BSD
- PicoBSD
- TrustedBSD

4.1.3. GNU/Linux

GNU/Linux[37][45] es una implementación de libre distribución UNIX para computadoras personales (PC), servidores, y estaciones de trabajo. Inicialmente desarrollado para las arquitecturas i386 fue creciendo soportando en la actualidad una diversidad mucho mayor de procesadores como por ejemplo i486, i686, AMD ,Cyrix, SPARC, DEC Alpha, PowerPC/PowerMac, y Mac/Amiga Motorola 680x0. Resumiendo sus principales características como sistema operativo podemos mencionar que es multitarea, multiusuario, multiplataforma y multiprocesador.

De manera similar a como surge BSD surge la comunidad GNU, a principios de la década del 80 el software se había vuelto propietario evitando y prohibiendo el acceso a los usuarios del código fuente, impidiendo de esta manera la cooperación entre los distintos desarrolladores para estudiar, compartir y mejorar el software existente. Este hecho hizo que Richard Stallman, un científico que venía trabajando en el MIT desde los 70 con software libre, en el año 1983 ideara la idea del movimiento GNU cuyo objetivo inicial fue desarrollar software que pudiera reemplazar el software propietario existente. La sigla GNU, no es una sigla en si, si no que es como si fuera una broma significa **GNU is Not Unix**.

El objetivo de GNU entonces es desarrollar un sistema operativo y un conjunto de aplicaciones para correr sobre el que fueran compatibles con Unix pero distribuidas como software libre. Comenzando con este proceso Stallman llego a la conclusión de que sería un enorme trabajo desarrollar un sistema completo desde cero por lo cual comenzó a incorporar a su proyecto otros softwares libres que estuvieran circulando y trabajando sobre ellos.

Con la necesidad de recaudar fondos para soportar el movimiento GNU en el año 1985 se funda la Free Software Foundation (FSF), la cual junta dinero a través de donaciones, venta de merchandising, manuales, copias de CDs y algún otro servicio.

Siguiendo con las necesidades apareció la urgencia de proteger el código generado evitando que alguien abusará de las libertades y por ejemplo se le ocurriera tomar un código fuente mejorarlo y cerrarlo, para cubrir esto aparece la licencia GPL.

Hacia el año 1990 la comunidad tenia una gran cantidad de software desarrollado y estaba casi listo para independizarse completamente de UNIX (vale la pena aclarar que casi todo el desarrollo inicial de GNU se hizo sobre Unix ya que es la plataforma a la que se quería reemplazar), lo único realmente importante que faltaba era un kernel. El kernel elegido inicialmente se lo llamaba Alix y luego se renombro a GNU HURD y estaba construido sobre el proyecto Mach, un microkernel desarrollado en la Universidad Carnegie Mellon y luego en la Universidad de Utah.

Como el GNU HURD no estaba suficientemente maduro para pasar a producción aparece en escena en 1991 Linus Torvalds, que desarrolla un kernel compatible con Unix el cual denomina Linux. Y así, en el año 1992 combinando el kernel Linux con el software GNU aparece el primer sistema GNU/Linux libre completo para reemplazar Unix. Se estima que hoy, varios años después hay varios millones de usuarios de Linux.

En cuanto a la comunidad y al Kernel el desarrollo continúa, pero no es sólo Linus el que mantiene el Kernel, a estas alturas el principal autor es la red Internet, desde donde un gigantesco grupo de programadores y usuarios aportan su tiempo y ayuda, tanto al núcleo como al resto de las aplicaciones. La FSF continúa con el proyecto GNU desarrollando otras aplicaciones que todavía no tienen su reemplazo libre.

Las distribuciones

En el ambiente GNU/Linux hay un concepto que es la distribución, que no es más que una forma de empaquetar código de aplicaciones GNU con un kernel Linux para poder distribuirlo.

Las principales o más conocidas[46]:

- **RedHat**: una de las más importantes, paso a ser comercial, las últimas versiones son pagas.
- **Fedora**: la versión gratuita de Red Hat, apareció cuando RedHat paso a ser comercial.
- **Suse**: De la misma forma que Red Hat se transformó en propietario al ser adquirida por Novell, y lanzo la versión libre OpenSuse.
- **Debian**: Una gran distribución completamente libre y sin ánimo de lucro, la distribución por excelencia de la GNU. Hay muchas distribuciones que basadas en ella, por ejemplo Lihuen GNU Linux.
- **Ubuntu**: Basada en Debian es una distribución con muchos seguidores ya que esta pensada para usuarios nuevos en el mundo GNU/Linux.
- **Gentoo**: Distribución basada en código fuente, hay que compilar todo para nuestra PC, lo que nos brinda mayor performance pero mayores tiempos para concluir con la instalación.
- **Slackware**: La distribución más antigua que aún tiene vigencia. Pensada para usuarios intermedios-avanzados.
- **Mandriva**: Surgió como resultado de la fusión de la distribución francesa Mandrake Linux y la brasileña Conectiva Linux, esta orientada a usuarios nuevos.
- **Knoppix**: basada en Debian, fue muy importante al introducir la idea del LIVE CD.
- **CentOS**: es un clon a nivel binario de la distribución Red Hat Enterprise Linux, compilado por voluntarios a partir del código fuente liberado por Red Hat.
- **LFS:** Linux From Scratch, la idea es que si no te gusta ninguna de las distribuciones existentes se pueda crear una desde cero. LFS te proporciona instrucciones y ayuda para instalar Linux partiendo directamente del código fuente de cada uno de sus paquetes.

Ver los logos de las más populares en imagen 4.2.

Podemos decir que en general las distintas distribuciones están relacionadas, ya que muchas veces una distribución esta basada en otra, o se inicio a partir de otra y hoy ya tiene un camino diferente, el concepto que se persigue es “no reinventar la rueda” sino reutilizar la experiencia y los logros de los demás. A pesar de ello podemos enumerar algunos aspectos que las diferencian:

- El instalador, hay instaladores tradicionales, se bootea de un CD, un DVD o un PENDRIVE y se instala, y hay versiones live, que permiten utilizar el sistema sin necesidad de instalarlo en el disco rígido, y solo instalarlo en caso de ser necesario.
- La estabilidad, hay distribuciones más conservadoras, que sólo traen software estable con un buen tiempo de pruebas y hay otras más audaces que traen la última versión de las utilidades, aunque no hayan sido testeadas adecuadamente.
- La forma en que se distribuyen las aplicaciones, algunas vienen con el código pre-compilado listo para usar y en otras hay que compilar el fuente de la aplicación para poder utilizarla.
- La licencia, hay algunas que sólo usan código GNU y hay otras que utilizan código GNU, propietario y código libre pero que necesita de una parte de soft propietario para funcionar, por ejemplo del firmware de algún dispositivo o el codec de un formato de compresión de video.
- La finalidad, dependiendo del objetivo para el cual esta diseñada la distribución será el conjunto de aplicaciones que trae, por ejemplo no es lo mismo una distribución orientada a educación que una orientada a Video Juegos.
- El sistema de empaquetamiento que maneja, facilitando la instalación, mantenimiento y desinstalación de aplicaciones.

Ver la inter-relación entre las distintas distribución en imagen 4.3

Figura 4.2: Logos de las principales distribuciones

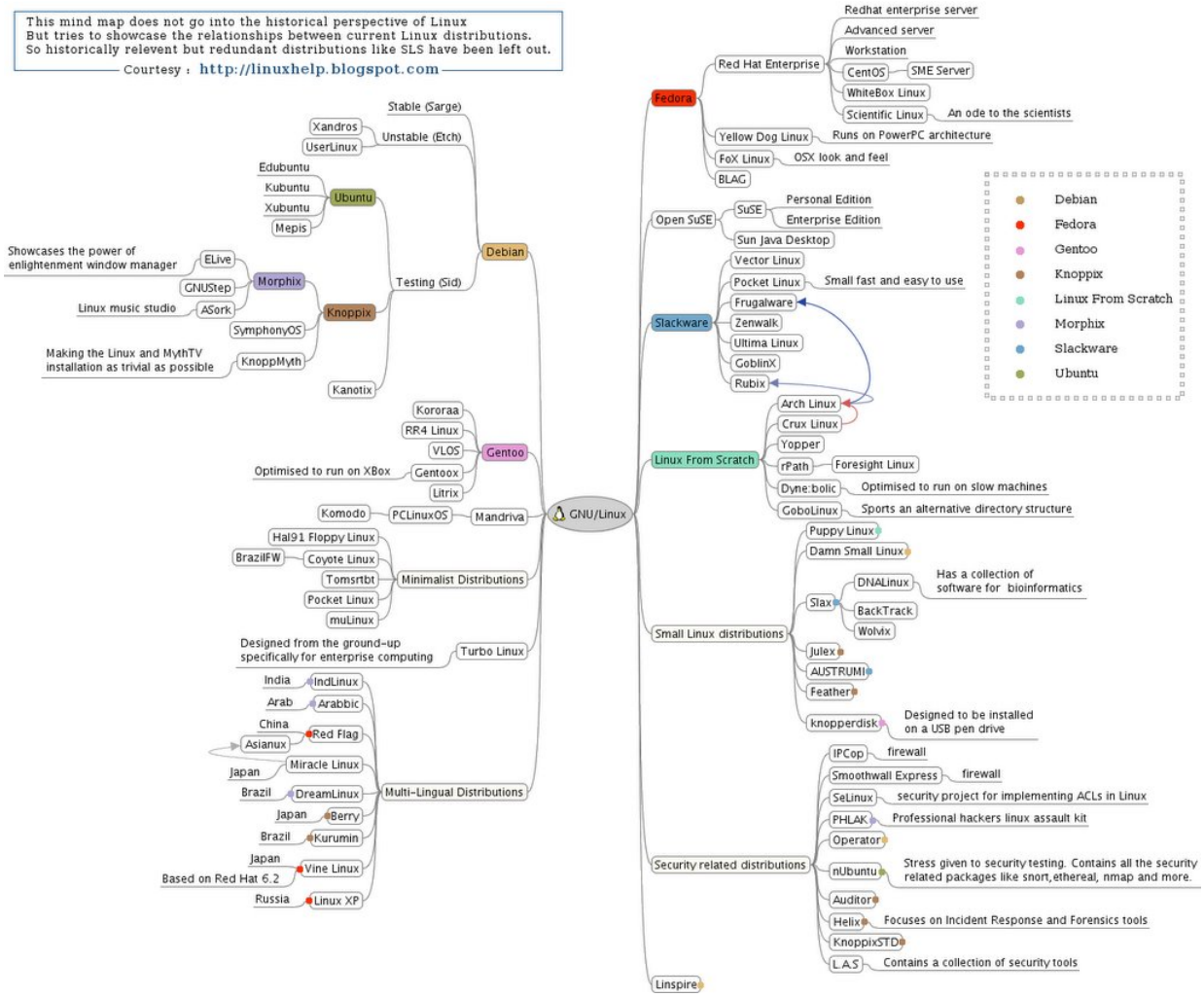

Figura 4.3: Mapa de las distribuciones GNU/Linux y su interrelación

4.1.4. El elegido

A la hora de elegir entre ellas nos encontramos con que sobre ambas plataformas existe una cantidad de aplicaciones similar y en general mucho del software GNU desarrollado para Linux está portado a BSD y viceversa, o sea que la mayoría del software libre disponible lo podemos correr tanto en BSD como en GNU/Linux.

Pasando a otros criterios de elección, analizamos el idioma, la comunidad que los soporta y la facilidad de instalación. En base a estos criterios no podemos decir qué rama elegir, ya que están empatadas debido a que las dos opciones tienen un estado similar en estos aspectos. Ambas se distribuyen en diversos idiomas, la facilidad de instalación es similar (sobre todo entre algunas versiones de cada rama) y la comunidades son enormes, tanto la de *BSD como la de *GNU/Linux.

Como todos los parámetros anteriores no nos alcanzan para justificar la elección de uno u otro nos basaremos en la experiencia personal. A saber, ambos utilizamos día a día en nuestras estaciones de trabajo alguna versión de GNU/Linux, en las redes que administramos habitualmente usamos GNU/Linux para montar los servicios, hemos dictado y tomado diversos cursos utilizando GNU/linux, y en general en todas las asignaturas de la Facultad en las que participamos, tanto de grado como postgrado, inculcamos y fomentamos el uso de GNU/Linux siempre que esté a nuestro alcance. Mientras que si bien sobre *BSD hemos realizado diversas pruebas

e instalaciones la experiencia no es la misma, ni en cantidad ni en satisfacción.

Entonces, tomando como parámetros las experiencias personales, elegimos la rama GNU/Linux. A partir de este punto debemos elegir entre las muchas opciones existentes la distribución o las distribuciones de de GNU/Linux que más nos gusten para realizar el trabajo, tratando de elegir la más adecuada para cada sección del trabajo.

Aquí nos orientamos a Debian GNU/Linux[47][1] para el servidor que albergará el desarrollo, y a alguna distribución basada en ella, pero orientada a Workstation para utilizar como estación de desarrollo, como por ejemplo Lihuen GNU/Linux[2]. Estas elecciones las hacemos basándonos nuevamente en la experiencia previa, ya que conocemos las facilidades que nos proveen, tanto para la instalación como para conseguir soporte en la comunidad. Las instalaciones son simples ya que Debian utiliza un sistema de empaquetamiento del software que simplifica el proceso. Y, en cuanto al soporte Debian tiene una enorme comunidad global a la que recurrir y, además tenemos la posibilidad de convocar al grupo de soporte de Lihuen GNU/Linux en el ámbito local de nuestra facultad.

4.2. El firewall

4.2.1. PF: The OpenBSD Packet Filter

Packet Filter[48], conocido como PF es el firewall utilizado en los sistemas OPENBSDs para realizar filtrado de tráfico TCP/IP y network address translation (NAT)[49].

Entre sus numerosas funcionalidades podemos citar al control de tráfico que realiza como la más importante ya que permite priorización de tráfico y controles de ancho de banda para lograr que se pueda compartir una conexión adecuadamente entre muchos clientes.

PF apareció en el kernel de OpenBSD desde OpenBSD 3.0, portándose luego a las otras distribuciones basadas en BSD, como por ejemplo a FreeBSD en el año 2004 y NetBSD desde la versión 3.0.

En este trabajo vamos a interrumpir el análisis de PF en este punto ya que al ser GNU/Linux el ambiente escogido para el desarrollo no tiene mucho sentido profundizar en PF.

4.2.2. Netfilter/IPtables

Netfilter[50] es un framework disponible en el kernel Linux que permite interceptar y manipular paquetes de red. Dicho framework permite realizar el manejo de paquetes en diferentes estados del procesamiento.

El componente más popular construido sobre **Netfilter** es **IPtables**, una herramientas de firewall que permite no solamente filtrar paquetes, sino también realizar traducción de direcciones de red (NAT) para IPv4 o mantener registros de log. **IPtables** es un software disponible en prácticamente todas las distribuciones de Linux actuales.

Antes de iptables, los programas más usados para crear un firewall en Linux eran **ipchains** en el kernel Linux 2.2 e **ipfwadm** en el kernel Linux 2.0, que a su vez se

[1]Debian GNU/Linux: una de las mayores distribuciones de GNU/Linux, http://www.debian.org

[2]Lihuen GNU/Linux: distribución GNU/Linux basada en Debian desarrollada en la Facultad de Informática de la UNLP, http://www.lihuen.info.unlp.edu.ar

basaba en **ipfw** de BSD. Tanto **ipchains** como **ipfwadm** alteran el código de red para poder manipular los paquetes, ya que no existía un framework general para el manejo de paquetes hasta la aparición de **netfilter**. **IPtables** mantiene la idea básica introducida en Linux con **ipfwadm**: listas de reglas en las que se especifica qué machear dentro de un paquete y qué hacer con ese paquete. **IPchains** agrega el concepto de cadenas de reglas (chains) e **IPtables** extendió esto a la idea de tablas: se consultaba una tabla para decidir si había que natear un paquete, y se consultaba otra para decidir como filtrar un paquete. Adicionalmente, se modificaron los tres puntos en los que se realiza el filtrado en el viaje de un paquete, de modo que un paquete pase sólo por un punto de filtrado.

Mientras que **ipchains** e **ipfwadm** combinan filtrado de paquetes y NAT (específicamente tres tipos de NAT, llamados masquerading o enmascaramiento de IP, port forwarding o redireccionamiento de puertos, y redirection o redirección), **netfilter** hace posible por su parte separar las operaciones sobre los paquetes en tres partes: packet filtering (filtrado de paquetes), connection tracking (seguimiento de conexiones) y Network Address Translation (NAT o traducción de direcciones de red). Cada parte se conecta a las herramientas de **netfilter** en diferentes puntos para acceder a los paquetes. Los subsistemas de seguimiento de conexiones y NAT son más generales y poderosos que los que realizaban **ipchains** e **ipfwadm**.

Esta división permite a **iptables**, a su vez, usar la información que la capa de seguimiento de conexiones ha determinado acerca del paquete: esta información estaba antes asociada a NAT. Esto hace a **iptables** superior a **ipchains**, ya que tiene la habilidad de monitorizar el estado de una conexión y redirigir, modificar o detener los paquetes de datos basados en el estado de la conexión y no solamente por el origen, destino o contenido del paquete. Un firewall que utilice **iptables** de este modo se llama firewall stateful, contrario a **ipchains** que solo permite crear un firewall stateless (excepto en casos muy limitados). Podemos decir entonces que **ipchains** no está al tanto del contexto completo en el cual un paquete surge, mientras que **iptables** sí y por lo tanto **iptables** puede tomar mejores decisiones sobre el futuro de los paquetes y las conexiones.

Tablas

Hay tres tablas ya incorporadas, cada una de las cuales contiene ciertas cadenas predefinidas. Es posible crear nuevas tablas mediante módulos de extensión. El administrador puede crear y eliminar cadenas definidas por usuarios dentro de cualquier tabla. Inicialmente, todas las cadenas están vacías y tienen una política de destino que permite que todos los paquetes pasen sin ser bloqueados o alterados. Ver imagen 4.4.

Filter table (Tabla de filtros)

Esta tabla es la responsable del filtrado (es decir, de bloquear o permitir que un paquete continúe su camino). Todos los paquetes pasan a través de la tabla de filtros. Contiene las siguientes cadenas predefinidas y cualquier paquete pasará por una de ellas:

- INPUT chain (Cadena de ENTRADA) : Todos los paquetes destinados a este sistema atraviesan esta cadena (y por esto se la llama algunas veces LOCAL INPUT o ENTRADA LOCAL).

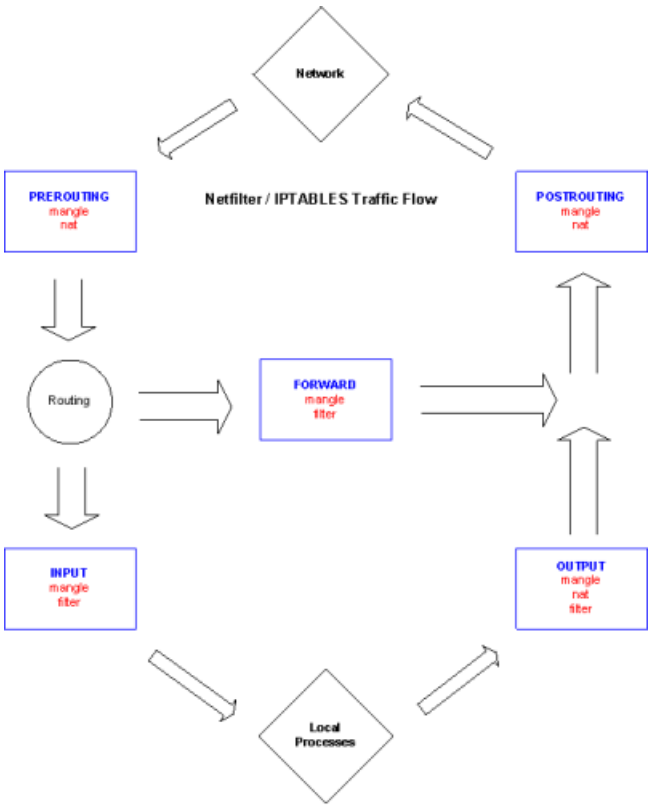

Figura 4.4: Tablas de netfilter/iptables

- OUTPUT chain (Cadena de SALIDA) : Todos los paquetes creados por este sistema atraviesan esta cadena (a la que también se la conoce como LOCAL OUTPUT o SALIDA LOCAL).
- FORWARD chain (Cadena de REDIRECCIÓN) : Todos los paquetes que meramente pasan por este sistema para ser encaminados a su destino recorren esta cadena.

Nat table (Tabla de traducción de direcciones de red)

Esta tabla es la responsable de configurar las reglas de reescritura de direcciones o de puertos de los paquetes. El primer paquete de cualquier conexión pasa a través de esta tabla; los veredictos determinan como van a reescribirse todos los paquetes de esa conexión. Contiene las siguientes cadenas predefinidas:

- PREROUTING chain (Cadena de PRERUTEO) : Los paquetes entrantes pasan a través de esta cadena antes de que se consulte la tabla de ruteo local, principalmente para DNAT (destination-NAT o traducción de direcciones de red de destino).
- POSTROUTING chain (Cadena de POSRUTEO) : Los paquetes salientes pasan por esta cadena después de haberse tomado la decisión del ruteo, principalmente para SNAT (source-NAT o traducción de direcciones de red de origen).
- OUTPUT chain (Cadena de SALIDA) : Permite hacer un DNAT limitado en paquetes generados localmente.

Mangle table (Tabla de destrozo)

Esta tabla es la responsable de ajustar las opciones de los paquetes, como por ejemplo la calidad de servicio. Todos los paquetes pasan por esta tabla. Debido a que está diseñada para efectos avanzados, contiene todas las cadenas predefinidas posibles:

- PREROUTING chain (Cadena de PRERUTEO) : Todos los paquetes que logran entrar a este sistema, antes de que el ruteo decida si el paquete debe ser reenviado (cadena de REENVÍO) o si tiene destino local (cadena de ENTRADA).
- INPUT chain (Cadena de ENTRADA) : Todos los paquetes destinados para este sistema pasan a través de esta cadena.
- FORWARD chain (Cadena de REDIRECCIÓN) : Todos los paquetes que exactamente pasan por este sistema pasan a través de esta cadena.
- OUTPUT chain (Cadena de SALIDA) : Todos los paquetes creados en este sistema pasan a través de esta cadena.
- POSTROUTING chain (Cadena de POSRUTEO) : Todos los paquetes que abandonan este sistema pasan a través de esta cadena.

Además de las cadenas ya incorporadas, el usuario puede crear todas las cadenas definidas por el usuario que quiera dentro de cada tabla, las cuales permiten agrupar las reglas en forma lógica.

Cada cadena contiene una lista de reglas. Cuando un paquete se envía a una cadena, se lo compara, en orden, contra cada regla en la cadena. La regla especifica qué propiedades debe tener el paquete para que la regla sea aplicada, como número de puerto o dirección IP. Si la regla no lo machea, el procesamiento continúa con la regla siguiente. Si la regla, por el contrario, machea el paquete, las instrucciones de destino de las reglas se siguen (y cualquier otro procesamiento de la cadena normalmente se aborta). Algunas propiedades de los paquetes solo pueden examinarse en ciertas cadenas (por ejemplo, la interfaz de red de salida no es válida en la cadena de ENTRADA). Algunos destinos sólo pueden usarse en ciertas cadenas y/o en ciertas tablas (por ejemplo, el destino SNAT sólo puede usarse en la cadena de POSRUTEO de la tabla de traducción de direcciones de red).

¿Y qué es un Filtro de Paquetes?

Un filtro de paquetes es un software que examina la cabecera de los paquetes según van pasando, y decide la suerte del paquete completo. Podría decidir descartarlo (DROP) (esto es, como si nunca lo hubiera recibido), aceptarlo (ACCEPT) (dejar que pase), o cosas más complicadas.

¿Cómo pasan los paquetes por los filtros?

El kernel empieza con tres listas de reglas en la tabla de "filtros"; estas listas se llaman cadenas del firewall o sencillamente cadenas. Se llaman INPUT, OUTPUT y FORWARD. Ver imagen 4.5

Cuando un paquete alcanza un círculo en el diagrama, se examina esa cadena para decidir la suerte del paquete. Si la cadena dice que hay que descartar

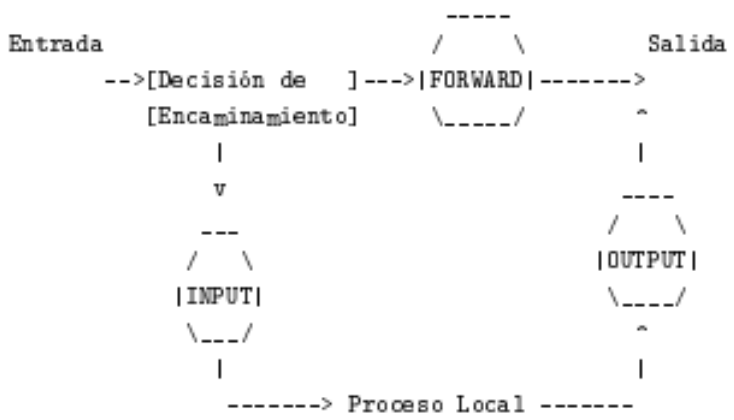

Figura 4.5: Filtros de netfilter/iptables

(DROP) el paquete, se lo desecha ahí mismo, pero si la cadena dice que hay aceptarlo (ACCEPT), continúa su camino por el diagrama. Una cadena es una lista de reglas. Cada regla dice "si el paquete se parece a esto, entonces esto es lo que hay que hacer con él". Si la regla no se ajusta al paquete, entonces se consulta la siguiente regla en la lista. Al final, si no hay más reglas por consultar, el núcleo mira la política de la cadena para decidir qué hacer. En un sistema consciente de la seguridad, esta política suele decirle al núcleo que descarte (DROP) el paquete.

1. Cuando llega un paquete por la interfaz de red el kernel mira primero su destino: a esto se le llama "encaminamiento" (routing).

2. Si está destinado a esa misma máquina, el paquete entra en el diagrama hacia la cadena INPUT. Si pasa de aquí, cualquier proceso que esté esperando por el paquete, lo recibirá.

3. En caso contrario, si el kernel no tiene las capacidades de reenvío activadas (forwarding), o no sabe hacia dónde reenviar el paquete, se descarta el paquete. Si está activado el reenvío, y el paquete está destinado a otra interfaz de red (si tenemos otra), entonces el paquete pasa directamente a la cadena FORWARD de nuestro diagrama. Si es aceptado, entonces saldrá de la máquina.

4. Finalmente, si un programa que se ejecuta en la máquina puede enviar paquetes de red. Estos paquetes pasan por la cadena OUTPUT de forma inmediata: si los acepta (ACCEPT), entonces el paquete continúa hacia afuera, dirigido a la interfaz a la que estuviera destinada.

4.2.3. Conclusión

Al ser GNU/Linux el ambiente escogido nos orientamos directamente a estudiar y utilizar para nuestro desarrollo el firewall NETFILTER/IPTABLES.

4.3. El analizador de vulnerabilidades

4.3.1. Nessus

Nessus[51][52] es un programa de escaneo de vulnerabilidades que corre en diversos sistemas operativos. Nessus es un software propietario, sólo es gratuito para uso

personal. Su objetivo es detectar las vulnerabilidades potenciales en los sistemas escaneados. En operación normal, Nessus comienza escaneando los puertos con alguno de sus escaneadores de puertos, e incluso puede utilizar NMAP[53][54], para buscar puertos abiertos y después intentar varios exploits para atacarlo. Las pruebas de vulnerabilidad, disponibles como una larga lista de plugins, son escritos en NASL[55] (Nessus Attack Scripting Language, Lenguaje de Scripting de Ataque Nessus por sus siglas en inglés).

¿Qué busca Nessus?

Básicamente, lo que Nessus busca detectar es:

- Vulnerabilidades que permitan a un atacante remoto controlar o acceder a información sensible de un sistema.
- Malas configuraciones.
- Sistemas desactualizados, sin sus últimos parches aplicados.
- Passwords por defecto, testea la ausencia de passwords o el uso de los más comunes, e incluso puede invocar una herramienta externa para lanzar un ataque por diccionario.
- Provocar un DoD contra el stack TCP/IP utilizando paquetes mal formados.

Algunas de las pruebas de vulnerabilidades de Nessus pueden causar que los servicios o sistemas operativos se corrompan y caigan. Hay que tener cuidado con los plugins que se utilizan en un escaneo.

Nessus es el escaneador de vulnerabilidades más popular. Se estima que es utilizado por más de 75.000 organizaciones alrededor del mundo. Se encuentra en el primer puesto de las encuestas sobre herramientas de seguridad de los años 2000, 2003 y 2006 de SecTools.Org.

¿Qué tipo de licencia tiene?

Nessus fue un escaneador de vulnerabilidades más popular hasta que cerraron el código en 2005 y quitaron la versión gratis en 2008. Aunque aún está disponible una versión limitada, la licencia permite su uso sólo en redes hogareñas. Algunas personas evitan pagar violando la licencia, en cambio otros lo utilizan sólo con los plugins que provee la instalación. Para la mayoría de los usuarios, el costo ha pasado de 0 a 1200 dolares al año. A pesar de esto, Nessus sigue siendo el mejor escaneador de vulnerabilidad en los sistemas UNIX y está entre los mejores que corren en Windows. Nessus debe actualizarse constantemente, ya cuenta con más de 20.000 plugins.

4.3.2. OpenVAS

OpenVAS[56] (Open Vulnerability Assessment System) es una herramienta para el análisis de seguridad de una red. El núcleo es un servidor que cuenta con un conjunto de Test de Vulnerabilidades de Red (o NVTs - Network Vulnerability Tests) para detectar problemas de seguridad en aplicaciones y sistemas remotos.

Todos los productos OpenVAS son libres con licencia GNU General Public License (GNU GPL). OpenVAS se deriva del proyecto Nessus que se ha convertido en un producto propietario. Siendo Nessus, como lo mencionamoes antes, uno de los más reconocidos escaneadores de vulnerabilidades que existen en la actualidad.

OpenVAS consta de 5 partes

- **OpenVAS-Server**: Es el núcleo de OpenVAS. Contiene la funcionalidad para escanear una gran cantidad de objetivos a alta velocidad. Este consta de 3 módulos:
  - **OpenVAS-Libraries**: Librerías que contienen funcionalidad que es utilizada por el OpenVAS - Server.
  - **OpenVAS-LibNASL**: Los NVTs antes mencionados son escritos en el lenguaje "Nessus Attack Scripting Language" (NASL). Este módulo contiene la funcionalidad requerida para que el OpenVAS-Server interactúe con NASL.
  - **OpenVAS-Plugins**: Contiene la base de datos de las NVTs. Esta debe ser actualizada con frecuencia para obtener las últimas vulnerabilidades existentes.
- **OpenVAS-Client**: El cliente se conecta con el OpenVAS-Server, procesa los resultados de los escaneos y los muestra al usuario. El cliente puede conectarse a múltiples servidores.

Ver la estructura de OpenVas en imagen 4.6.

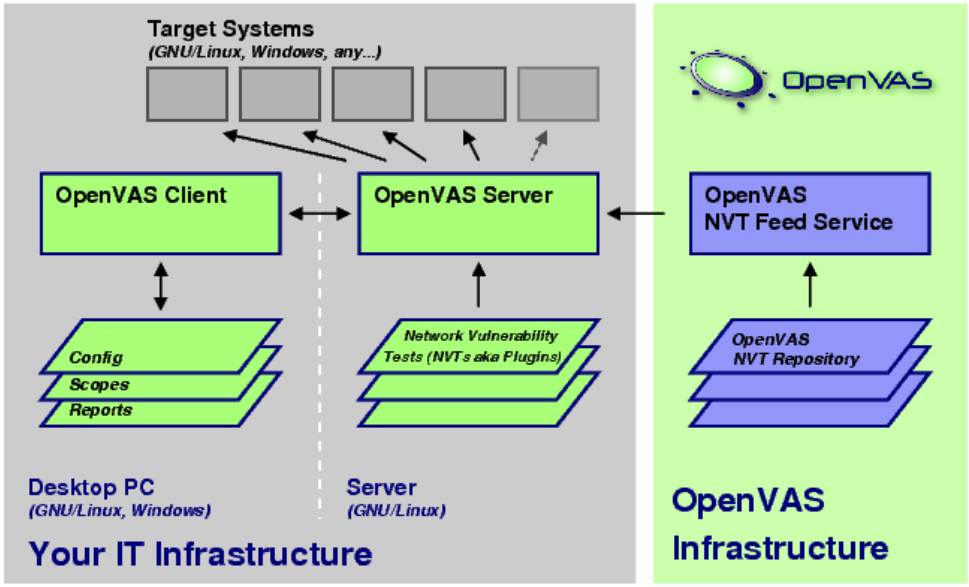

Figura 4.6: Estructura de OpenVAS

Hay que tener presente que los plugins que OpenVAS libera (en el modulo "openvas-plugins") son un conjunto básico de tests. El ciclo de actualizaciones de este conjunto base es demasiado largo en comparación con la ocurrencia de nuevas vulnerabilidades y sus respectivas NVTs. Los nuevos tests o cambios en los mismos son distribuidos mediante los llamados "feed services", que no son otra cosa que servidores de actualizaciones de NVTs. Si uno quiere testear la red contra las últimas amenazas o vulnerabilidades debemos suscribirnos a "feed services" confiables y reconocidos, que sean actualizados conforme vayan apareciendo las nuevas amenazas. El proyecto OpenVAS posee un "feed service" por si mismo: http://www.openvas.org/openvas-nvt-feed.html

4.3.3. Conclusión

Nos quedaremos con OpenVAS por ser el más desarrollado y aceptado en la comunidad de entre los escaneadores de vulnerabilidades existentes y que mantiene la licencia libre.

4.4. El IDS

4.4.1. BRO - IDS

Bro[57], cuyo nombre que proviene de BIG BROTHER (Gran Hermano) donde la idea es que todo lo que ocurre es visto por el que controla, es un sistema de detección de intrusiones de red (NIDS) para sistemas *UNIX y *BSD distribuido como open source que monitorea de forma pasiva el tráfico que circula por la red en busca de actividad sospechosa.

Para poder detectar intrusiones primero parsea el tráfico de red para extraer los datos de nivel de aplicación y después ejecuta analizadores orientados a eventos que compara los datos extraídos con patrones de actividad problemática. El análisis incluye detección de ataques específicos (tanto los que se detectan mediante firmas como los que se detectan gracias a los eventos que genera el tráfico) como actividades poco usuales como por ejemplo ciertos host conectándose a ciertos servicios o fallos en el intento de establecimiento de conexiones.

Bro utiliza un lenguaje especial para escribir las políticas que indicarán al sistema la forma de funcionar, esto permite ajustar la operación de BRO, tanto como para adaptarse a la forma en que las políticas del lugar donde esta instalado vayan cambiando como para detectar los nuevos patrones de ataques que vayan apareciendo. Si se detecta algo de interés puede instruirse a Bro a generar un log, generar una alerta en tiempo real al operador o a ejecutar alguna operación en el sistema, como por ejemplo terminar una conexión o bloquear en el firewall algún host malicioso al vuelo. Como los logs son configurables mediante las políticas definidas pueden realizarse con toda la información que consideremos necesaria, por lo que pueden utilizarse sus logs como una ayuda interesante para trabajos de forensia.

Además si hubiéramos capturado el tráfico Bro puede procesarlo nuevamente, o sea que podemos reprocesar tráfico que ya paso por la red utilizando nuevas políticas para detectar patrones que al momento de captura del tráfico no era reconocidos como ataques pero hoy si lo son lo cual también es sumamente útil para trabajos de forensia. Por ejemplo si seguimos en la captura el comportamiento de un virus podríamos saber que hosts fueron contactados por ese malware [3].

El objetivo de análisis de Bro son las redes de alto volumen de tráfico como redes Gbps. Esta pensado para ejecutarse en hardware tipo PC, de las que se consiguen hoy día cuando vamos a comprar una PC de escritorio por lo que no es necesaria una gran inversión en infraestructura.

Tenemos que tener en cuenta que Bro nació como una plataforma de investigación y que esta diseñado para ser un sistema de detección de intrusiones muy flexible y altamente configurable, por lo que no esta pensado para funcionar como una solución de las que se conoce como “out of the box” que funcionan ni bien se instala, sino que requiere mucha configuración inicial y mantenimiento posterior. Por estos

[3]Malware (del inglés malicious software), también llamado badware, software malicioso o software malintencionado es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora sin el consentimiento de su propietario.

motivos se puede decir que el usuario típico de Bro son usuarios expertos de sistemas *Unix o *BSD que estén a la altura de poder modificar las políticas para funcionar adecuadamente.

Bro es open source, si bien no esta protegido por GPL esta distribuido con una variante como software libre por la University of California, mediante el Lawrence Berkeley National Laboratory.

¿Cómo funciona BRO-IDS?

- A nivel de red escucha pasivamente el tráfico que pasa por la interfase de la PC y envía hacia arriba una copia de todo el tráfico de red.
- Luego el kernel filtra las grandes cantidades de tráfico obtenidas antes, este filtrado se realiza a través librerías de captura de paquetes libpcap.
- Siguiendo el proceso mediante el "Event engine" se traduce los streams filtrados con las libpcap en eventos de alto nivel que reflejan la actividad de la red. Por ejemplo: connection_attempt, http_reply, user_logged_in. Actualmente hay mas de 300 tipos diferentes.
- Luego el "Policy script" procesa los eventos incorporando contexto de eventos anteriores y las políticas particulares del sitio. Una vez que los eventos se procesan a través de las políticas en general se efectúa alguna acción como puede ser: grabar a disco, generar alertas mediante syslog, enviar eventos a otro Bro o ejecutar algún programa o script.

En la figura 4.7 vemos como funciona BRO-IDS desde la red hasta la aplicación de la política.

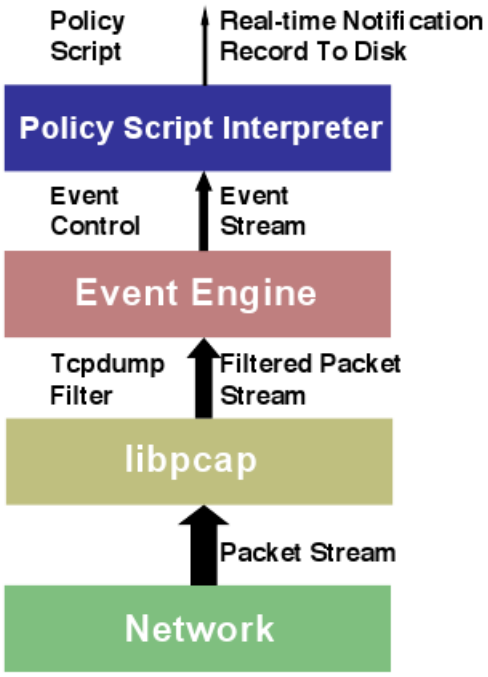

Figura 4.7: BRO-IDS

El contenido de el archivo **myhandlers.bro** mostrado a continuación es un ejemplo de política de Bro en la que lo que se hace es definir que hacer cuando se detecta

un establecimiento de conexion TCP sin diferenciar que tipo de conexión se este estableciendo. Lo que se hace en esta política es al detectar el establecimiento de una conexión imprimir la IP origen y la IP destino.

Contenido de myhandlers.bro:

```
    event connection_established(c: connection)
{
local orig = cidorig_h;
local resp = cidresp_h;
print orig, resp;
}
```

Las políticas de BRO están escritas en un lenguaje de scripting propio del IDS, el cual es similar a C, pero orientado a red.[58][59]

4.4.2. SNORT

En general cuando se habla de IDS en el ambiente inmediatamente se relaciona con Snort[60] ya que es uno de los NIDS/NIPS open source mas distribuidos en la comunidad.

Se lo reconoce capaz de realizar tanto análisis de tráfico en tiempo real como de logueo de paquetes, siempre en redes IP.

Puede realizar análisis de protocolos, búsqueda y evaluación de contenidos en los paquetes y puede ser utilizado para detectar una variedad de ataques, como son buffer overflows, escaneos de puertos incluyendo los que son en modo stealth, ataques de CGI, conexiones SMB, intentos de OS fingerprinting, y muchos más.

¿Cómo funciona SNORT?

Snort puede ser configurado de distintas maneras, pudiendo funcionar:

- En modo Sniffer, simplemente lee los paquetes de la red y los muestra al usuario en la consola.
- En modo Packet Logger, guarda los paquetes que escucha en el disco.
- En modo Network Intrusion Detection System (NIDS), en este modo Snort analiza el tráfico que pasa en busca de patrones que macheen con las reglas definidas por el usuario y realiza acciones basándose en lo que detecta.
- En modo Inline, en este modo el soft obtiene los paquetes directamente desde el firewall (Iptables) en vez de obtenerlos desde la libpcap como sucede en los 3 modos anteriores, analizando el tráfico de la misma forma que en modo NIDS. En base a las reglas lo reinyecta en el iptables haciendo que el firewall o bien lo descarte o bien lo deje pasar.

Componentes internos: Obteniendo y procesando los paquetes

Para observar el funcionamiento de Snort y sus componentes internas de manera gráfica podemos utilizar la figura 4.8.

Como Snort no tiene la posibilidad de obtener los paquetes desde la interfase de red directamente con un mecanismo propio, el soft se basa en el uso de las librerías **libpcap**[61], las que, al ser multiplataforma hacen que Snort también pueda serlo.

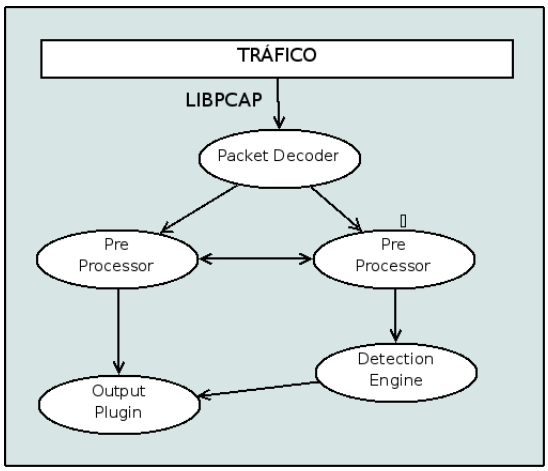

Figura 4.8: Funcionamiento de Snort

Entonces podemos decir que la responsabilidad de obtener los paquetes desde la placa de red y pasarlos sin alterar a Snort es de las libpcap.

A continuación aparece en escena el **Packet Decoder**, una vez obtenidos los paquetes se pasan a través de una serie de rutinas de decodificación, con las que se obtienen datos necesarios de los distintos protocolos de las distintas capas del stack de red. Se comienza en los niveles mas bajos de los protocolos de enlace, decodificando cada protocolo que encuentra mientras va subiendo entre las capas por ejemplo protocolos de capa de transporte, obteniendo información como los puertos TCP y UDP. A medida que los paquetes atraviesan los distintos decoders, una estructura se va llenando con los datos obtenidos. Una vez completa esta estructura esta lista para pasarse a los preprocesadores.

Los **preprocesadores** se pueden dividir en dos categorías, según como se utilizan:

- Para buscar actividad sospechosa detectando ataques que no pueden serlo basándose en firmas.
- Para modificar los paquetes de manera de poder pasarlos luego al motor de detección y que éste sea capaz de interpretarlos. En resumen lo que se hace es normalizar el tráfico de manera que el motor de detección pueda luego procesarlo buscando algún match con su base de firmas.

Ambas categorías buscan derrotar las técnicas utilizadas por los atacantes para evitar el éxito de Snort en la detección.

Hay varios, pero citamos dos ejemplos de preprocesadores:

- **FRAG2**: es el preprocesador para evitar técnicas de evasión de IDS mediante fragmentación de paquetes.
- **stream4**: es el preprocesador para mantener el estado de las conexiones TCP.

El siguiente, es el componente principal de Snort, el **Detection Engine**. Tiene dos funciones esenciales para el funcionamiento del IDS:

- Parseo de reglas
- Detección de firmas

Lo primero que hace es generar las firmas que permiten la detección de ataques procesando una a una las reglas de Snort en cuanto se levanta el servicio. Concretamente lo que hace es generar una estructura de árbol que le va a servir para tratar de matchear el tráfico a partir de lo definido en la cabecera de la regla, cuando el tráfico analizado concuerda se analiza si además coincide con lo definido en las opciones. Una vez obtenidas las firmas desde las reglas, el motor de detección se encarga de pasar el tráfico a través de la estructura generada buscando matcheos.

Las **Reglas** de Snort respetan una estructura especial, se dividen en dos secciones:

- El encabezado o header de la regla
- Las opciones

En el encabezado encontraremos la información que indica las condiciones bajo las cuales se aplicará la regla. Se puede especificar protocolo, las direcciones IP (origen y destino), los puertos (origen y destino) y el tipo de acción.

En las opciones de la regla vamos a ver que empieza y termina con un paréntesis, y que contiene la firma actual, el nivel de prioridad y alguna documentación sobre el ataque.

Ejemplo de la regla para detectar un intento de OpenSSH CRC32 remote exploit:

alert tcp $EXTERNAL_NET any -> $HOME_NET 22

(msg:"EXPLOIT ssh CRC32 overflow /bin/sh"; flow:to_server,established; content:/bin/sh"; reference:bugtraq,2347; reference:cve,CVE-2001-0144; classtype: shellcode-detect; sid:1324; rev:3;)

Explicación: A partir de la regla de ejemplo Snort generará un alerta con todo el tráfico TCP cuya IP origen sea cualquiera que coincida con las definidas en la variable interna de Snort $EXTERNAL_NET (generalmente utilizada para las redes externas) desde cualquier puerto TCP origen con IP destino cualquiera de las definida en la variable $HOME_NET (usualmente se la setea con la red que se quiere proteger) hacia puerto TCP 22, habitualmente servicio de ssh, y que además sea una conexión establecida con flujo hacia el server y que en el contenido tenga el string "/bin/sh".

Este alerta tendrá como descripción "EXPLOIT ssh CRC32 overflow /bin/sh".

En modo IDS las reglas de Snort tienen cinco acciones posibles:

- **alert:** genera una alerta y loguea el paquete.
- **log:** loguea el paquete.
- **pass:** ignora el paquete
- **activate:** genera una alerta y activa una regla dinámica.
- **dinamyc:** permanece ociosa hasta ser activada por una regla activate, a partir de ese momento actúa como una regla del tipo log.

En modo IPS se agregan otras 3 acciones posibles:

- **drop:** instruye a iptables para que descarte el paquete y luego funciona como una regla log de Snort.
- **reject:** pide a iptables que descarte el paquete, lo loguea y luego interrumpe la conexión, en caso de ser TCP enviará un reset spoofeado a ambos extremos y en caso de ser UDP enviará ICMP del tipo port unreachable spoofeados a ambos hosts.
- **sdrop:** solo se instruye a iptables a dropear el paquete, nada se loguea.

Ejemplo funcionando como Snort In Line.

alert ip any any -> any any
(msg:"SHELLCODE x86 stealth NOOP"; sid:651;
content:"—EB 02 EB 02 EB 02—";
replace:"—24 00 99 DE 6C 3E—";)

Explicación: En esta regla se toman todos los paquetes IPS de cualquier ip y puerto origen que vayan a cualquier ip y puerto destino y si tienen en su contenido —EB 02 EB 02 EB 02— se altera el paquete reemplazando ese string por —24 00 99 DE 6C 3E— generando un alerta con el mensaje "SHELLCODE x86 stealth NOOP".

Como se puede deducir que al depender de reglas escritas, vamos a tener la necesidad de actualizar la base de reglas de manera regular de forma de ser capaces de detectar los nuevos ataques que vayan apareciendo. Si queremos estar al día con las últimas firmas disponibles aquí es donde se acaba el software libre, para poder hacerlo hay que pagar una suscripción anual. De todas formas, si nos registramos en el sitio del fabricante tenemos acceso a las firmas con un delay de unos 30 días desde que aparecen.

Por último en este resumen sobre el funcionamiento de Snort tenemos que mencionar los **Output Plugins** que no es mas ni menos que el mecanismo que Snort tiene para comunicarse con el usuario. Ya sea un usuario humano que quiera leer lo detectado por Snort como algún otro software diseñado para funcionar en base a las alertas o logs del IDS. Para esto se presentan distintas alternativas, incluyendo la generación de logs con distintos formatos y la posibilidad de escribir en bases de datos como Mysql o Postgresql. En la actualidad consta con 12 mecanismos distintos.

Si bien Snort es capaz de escribir directamente en una base de datos, esto no es recomendable como solución para instalaciones en producción ya que el delay que le puede producir generar una inserción en una base de datos puede derivar en la perdida de paquetes por el mecanismo de detección, lo que se recomienda en cambio es utilizar alguna de las soluciones de output en forma binaria. Para solucionar este inconveniente existen distintas alternativas de solución que consisten en softwares que independientemente de Snort se ocupan de estar atentos a la salida de este y una vez que el IDS produce algo el plugin generará la inserción en la base de datos, sin interferir en el funcionamiento de los mecanismos de detección.

Plugin para Snort - Barnyard2

Barnyard2[62] es un interprete open source para archivos de salida de Snort en formato binario unified2. Se usa para escribir en distintos formatos la salida binaria a disco del Snort manera eficiente y en un proceso independiente, evitando que Snort pierda tráfico de red. Uno de los formatos de salida que admite es a base de datos MySQL que es lo que necesita nuestra aplicación.

4.4.3. Conclusión

Bro es open source, permite ejecutarse con recursos de hardware mínimos para monitorear redes de altos volúmenes de tráfico, funciona correctamente sobre GNU Linux y presenta facilidades para interrelacionarse directamente con el firewall o con cualquier otro software con el que se quiera relacionar. Si bien Bro tiene la posibilidad de utilizar las reglas de Snort importándolas directamente tiene una gran contra, se necesita un usuario avanzado que este permanentemente manipulando las políticas para ser efectivo y la instalación no es del todo automática.

Por su parte Snort aún con la excepción de la necesidad de licencia para obtener las reglas más nuevas disponibles es también software libre, funciona correctamente sobre GNU/Linux, es muy simple de instalar sobre Debian y presenta facilidades para generar salidas que después puedan procesarse fácilmente con nuestro sistema.

En la elección de IDS no es tan natural como las decisiones anteriores ya que ambos cumplen con nuestros requisitos y si bien la idea es preparar a nuestro sistema para que pueda configurarse con cualquiera de las dos opciones, en un primer paso haremos que el sistema se integre con la alertas detectadas por Snort tomando como principal factor de valoración la facilidad de instalación y los plugins existentes para Snort como Barnyard2 que van a simplificar el desarrollo de nuestra solución.

Capítulo 5

El desarrollo

5.1. Resumen del Objetivo

como se ha descripto en el primer capítulo, el objetivo fue desarrollar un sistema que integre la funcionalidad de un Firewall, de un sistema de detección de intrusiones y de un analizador de vulnerabilidades. Concretamente se integraron los siguientes productos:

- El firewall de GNU/Linux Iptables
- El IDS Snort con su helper Barnyard2
- Y el analizador de vulnerabilidades OpenVAS

La integración tiene como objetivo facilitar la administración de una red del tipo DMZ, donde, de forma descentralizada, múltiples usuarios (que representan a los distintos administradores de los distintos servidores que se encuentren en ella) puedan:

- Hacer un seguimiento del estado en que se encuentran sus servidores.
- Solicitar habilitaciones y denegaciones de permisos para el tráfico desde y hacia sus servidores.
- Visualizar si existe tráfico sospechoso saliendo o llegando a sus equipos.

Toda esta gestión se realiza a través de una cómoda y simple interfaz web a la que el usuario del sistema puede acceder utilizando el navegador web que desee. En forma gráfica puede observarse en la figura 5.1.

5.2. ¿Cómo lo hicimos?

El software propuesto fue desarrollado con una arquitectura cliente servidor, utilizando productos de software libre. Concretamente mediante la utilización de GNU/Linux como sistema operativo del servidor donde esté alojada la aplicación, Apache como servidor Web, Mysql[63] como el motor de base de datos y Perl[64] como lenguaje de programación del lado del servidor y la utilización de cualquier navegador web como cliente para poder acceder al sistema.

Otro objetivo que se persiguió fue desarrollar un producto cuyo código fuente sea fácilmente legible, adaptable, modificable y mantenible por cualquier desarrollador

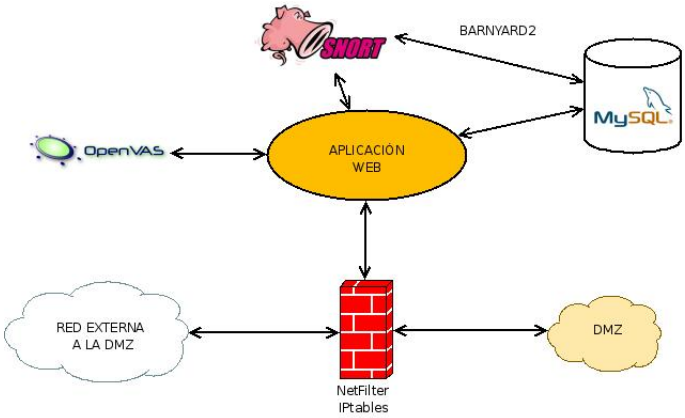

Figura 5.1: Integración de las componentes

que tenga un conocimiento mínimo en Perl y que así se lo proponga. Para cumplir con estas expectativas tomamos como premisa desarrollar respetando el modelo MVC[1][65], siguiendo el concepto de DRY[2][66], y según el paradigma de desarrollo con orientación a objetos. Estos temas son suficiente material de estudio para una tesina similar a la que estamos desarrollando, por lo tanto, dejaremos por el momento el análisis en profundidad de estos conceptos.

Estas metas nos plantearon una encrucijada, tuvimos que elegir como desarrollar. Las alternativas eran empezar el desarrollo utilizando perl en crudo directamente, sólo respetando lineamientos de desarrollo ad-hoc autoimpuestos y teniendo especial cuidado en respetarlos, o hacerlo apoyándonos en alguna plataforma de desarrollo que facilitara la tarea.

5.2.1. El framework

En los tiempos que corren y sobre todo cuando nos adentramos en el mundo del desarrollo en software libre nos encontramos con que, para la mayoría de los lenguajes de programación, existen frameworks de desarrollo, que no son ni más ni menos que herramientas que ayudan a los desarrolladores, a lograr los objetivos de legibilidad, adaptabilidad, modificabilidad y mantenibilidad planteados como premisas anteriormente. Con estas ideas es que decidimos adoptar la utilización de uno de ellos para nuestro trabajo.

Si hacemos una búsqueda rápida por Internet encontraremos varios frameworks de desarrollo, entre los más famosos y sólo para nombrar algunos están Ruby on Rails[67] para el lenguaje Ruby, Django[68] para Python y Symfony[69] o CakePHP[70] para Php. Para Perl existen varios también, pero de todos ellos el que mejor documentación y críticas tiene en la comunidad es Catalyst-Framework[71], por ello lo hemos elegido.

En el resultado final del desarrollo podemos juzgar la adopción de Catalyst como muy positiva, ya que además de que las 4 premisas declaradas se respetan, hay varios aspectos que se potenciaron como por ejemplo el uso de un ORM[3] que nos

[1]Model View Controller(fuente wikipedia):es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control

[2]Don't Repeat Yourself: principio que promueve la reducción de la duplicación del código, simplificando las modificaciones futuras

[3]ORM:es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, no necesariamente compatibles entre sí.

permitiría cambiar el motor de la base de datos que usa la aplicación sin mayores dificultades. Así como el uso de las ya incorporadas librerías de Javascript que facilitan el desarrollo.

Sin embargo hay que mencionar que la cantidad de tiempo y esfuerzo invertidos en el aprendizaje sobre Catalyst fueron considerables, ya que aprovechar el framework razonablemente hay que comprender y adaptarse a los lineamientos que impone.

5.3. ¿Qué es lo que hace?

Para comprender la idea de nuestro desarrollo tenemos que ser más detallistas en la descripción de las distintas secciones y funciones que tiene.

El sistema tiene 6 módulos distintos que trabajan de manera relacionada: Usuarios, Servidores, Tráfico, Escaneo de Vulnerabilidades, Monitoreo de Tráfico y Preferencias del sistema.

Dichos módulos se relacionan de la siguiente forma:

- El sistema tiene n usuarios.
- Cada usuario tiene de 0 a n servidores.
- Cada servidor posee:
    * 0 a n solicitudes sobre el tráfico.
    * 0 a n alertas de seguridad.
    * 0 a n escaneos de vulnerabilidades.

Las relaciones entre las distintas componentes en un nivel de abstracción y de manera muy simplificada se reflejan en la figura 5.2

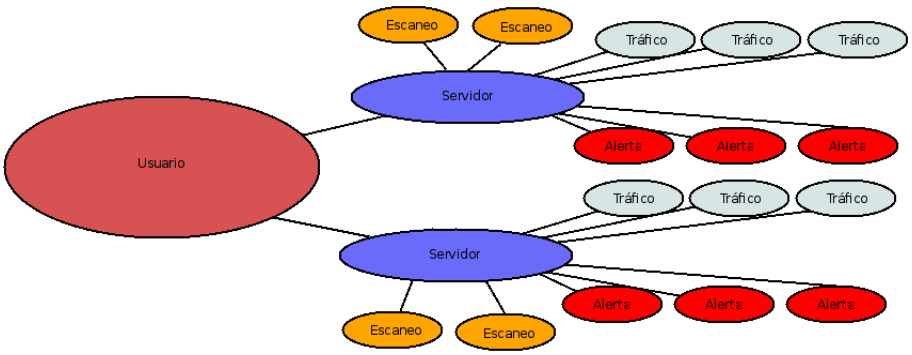

Figura 5.2: Relación entre las distintas componentes del sistema

También están las preferencias del sistema, que regulan todo el funcionamiento, tanto global como de cada una de las partes del soft, pero que no están relacionadas directamente.

A continuación describiremos detalladamente cada componente.

5.3.1. Usuarios

Perfiles

Para el funcionamiento del sistema vamos a tener distintos perfiles de usuarios, cada uno de los cuales tendrá distintas posibilidades dentro del sistema:

- **SuperUsuario**: es el super-usuario del sistema, tiene todos los permisos posibles y puede realizar cualquier acción de las disponibles.
- **Administrador**: es un perfil con menos privilegios que el superusuario, es el que usarían los administradores del sistema.
- **Usuario**: es el tipo de usuario que representa a los administradores de los distintos servidores o subredes de la red que protege el sistema.
- **Anónimo**: este tipo de usuario no puede realizar ninguna acción, sólo tendrá acceso a información de contacto y a la pantalla de login.

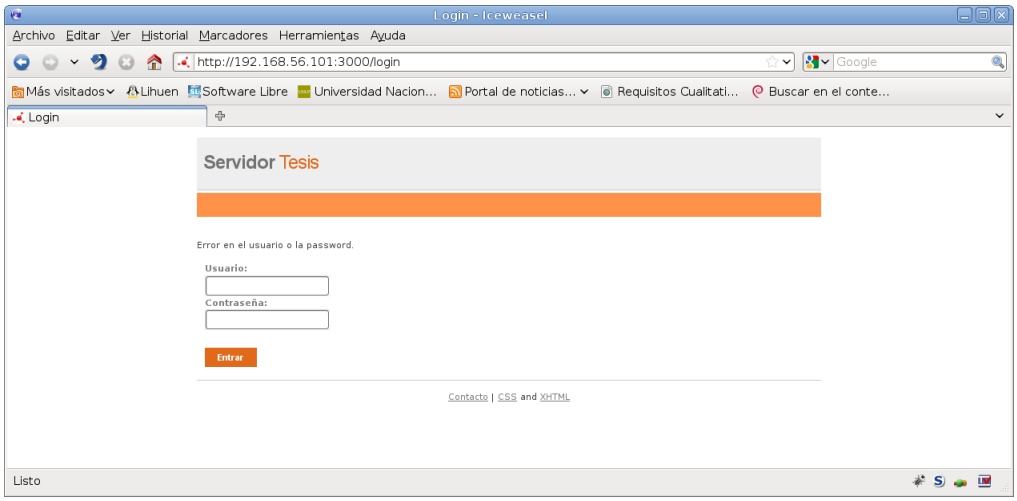

Figura 5.3: Pantalla de Login del sistema

ABM de Usuarios

Alta, baja y modificación de usuarios: es una función fundamental ya que como se mencionó antes el sistema no permitirá el acceso anónimo, uno siempre debe autenticarse, esto es porque la información que se maneja es sensible y puede comprometer la seguridad de la red.

El SuperUsuario puede dar de alta, baja o modificar usuarios de perfil Administrador o Cliente.

El Administrador sólo puede dar de alta, baja o modificar usuarios de perfil Cliente. Todos los perfiles tienen permiso de modificar sus propios datos, con excepción del rol. Ver figura 5.4.

5.3.2. Servidores

Alta, baja y modificación de servidores.

El alta de un servidor puede ser solicitada por un usuario Cliente, Administrador o SuperUsuario pero debe ser habilitado luego por un usuario con perfil Administrador o SuperUsuario para que se aplique el comportamiento definido para ese equipo.

Para aprobar este alta es deseable que el administrador verifique que el que solicita el servicio sea realmente responsable de ese equipo. Dicha comprobación excede a las funciones del sistema y será responsabilidad del usuario que lo habilite.

Figura 5.4: Pantalla de ABM de usuarios

Desde el momento que se habilita el servidor las reglas de tráfico, alertas y escaneos de vulnerabilidades asociados comienzan a ser tenidas en cuenta por el sistema. El servidor quedará asociado definitivamente al usuario que lo dio de alta.

La modificación y baja de los equipos podrá ser solicitada tanto por el usuario asociado al servidor como por un usuario de nivel superior.

Lista de servidores

Cada usuario puede tener N servidores, por lo que es necesario presentarle la lista de los servidores que tiene registrados. A partir de dicha lista puede acceder a las alertas registradas para cada servidor, a las reglas del firewall que afectan a cada servidor y a las auditorías de seguridad que se han realizado sobre cada servidor. Ver figura 5.5.

Figura 5.5: Lista de Servidores

5.3.3. Tráfico

ABM de solicitudes de habilitación y revocación de tráfico

Una vez que un cliente da de alta un servidor podrá realizar altas, bajas y modificaciones de solicitudes de permisos para tráfico desde o hacia ese servidor.

El alta de tráfico habilitado funciona de la siguiente manera, una vez que el Cliente realiza una solicitud de habilitación para determinado tráfico desde o hacia su servidor, ésta queda en un listado de tráfico entrante o tráfico saliente de acuerdo al carácter de la solicitud.

Como contrapartida a la solicitud de tráfico habilitado existe la posibilidad de solicitar bloqueo de tráfico hacia o desde el servidor.

Las solicitudes de habilitación/bloqueo de tráfico tienen distintos parámetros para configurar:

Si es entrante:

- Protocolo
- Puerto Destino
- IP Origen
- Máscara Origen
- Estado

Si es saliente:

- Protocolo
- Puerto Destino
- IP Destino
- Máscara Destino
- Estado

Ver figura 5.6.

Actualización de reglas

Para mantener las reglas actualizadas se debe programar en el firewall una tarea que genere un requerimiento al sistema web, el cual evalúa si es necesario reprocesar las reglas definidas para generar un nuevo listado o no.

Esta condición de regeneración de las reglas puede ocurrir por diversos motivos:

- Ante la modificación de los datos del servidor.
- Ante el alta, baja o modificación de una solicitud de bloqueo para tráfico desde o hacia ese servidor.
- Ante el alta, baja o modificación de una solicitud de autorización para tráfico desde o hacia ese servidor.
- Ante la habilitación o inhabilitación de estas mismas solicitudes.

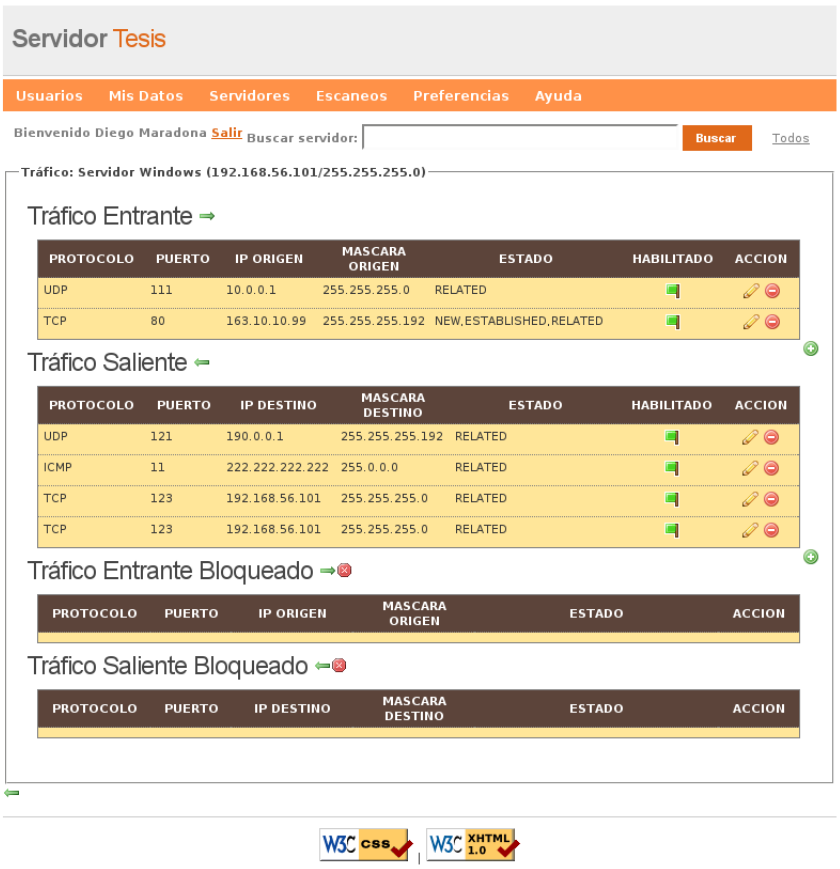

Figura 5.6: ABM de tráfico

- Ante la habilitación o inhabilitación de un servidor completo.

Si se genera un nuevo conjunto de reglas, el firewall las recibe a causa del requerimiento y las aplica inmediatamente. Esto se instrumenta mediante una tarea programada que consulta al sistema periódicamente, el intervalo varía de acuerdo a como se lo quiera configurar, por ejemplo: podría ser una vez por minuto.

Para comprender como funciona la integración con el firewall es vital aclarar algunos puntos:

- Las reglas que responden a las solicitudes de bloqueos de tráfico siempre van a aplicarse antes de las reglas para la aceptación de tráfico en el listado que se genera, así que hay que tener especial cuidado en como se escriben estos requerimientos.

- El sistema modificará exclusivamente las reglas de la cadena FORWARD de la tabla filter por lo que las políticas que afectan al tráfico destinado al equipo donde esté el firewall, deben ser configuradas por separado e independientemente de nuestra aplicación. Esto es, como mínimo las cadenas OUTPUT e INPUT de la tabla filter.

- El sistema genera reglas respetando una política restrictiva, es decir, que lo que no está expresamente permitido está prohibido por defecto.

- Nuestra aplicación no necesariamente debe correr en el mismo equipo que ejecuta el firewall.

- Como medida de seguridad al punto anterior se agregó una restricción para que las reglas para firewall sólo puedan ser solicitadas desde alguna dirección ipv4 en particular. Esta se debe configurar desde el listado de preferencias del sistema, alterando el valor de la variable **ip_firewall**.

5.3.4. Escaneos de vulnerabilidades

ABM de solicitudes de escaneo de vulnerabilidad

Una vez que un cliente da de alta un servidor podrá realizar altas, bajas y modificaciones de escaneos en busca de vulnerabilidades sobre ese equipo.

Al momento de solicitar escaneos el cliente podrá configurar la periodicidad de cada uno de ellos

- Única vez
- Diaria
- Semanal
- Mensual

Ver figura 5.7.

Figura 5.7: Alta de un escaneo para un servidor

¿Cómo se ejecutan?

Para ejecutar los escaneos en el momento que corresponda se disparan varias tareas cuya función es obtener los servidores a escanear en ese instante y ejecutar el cliente OpenVAS para que envíe al servidor OpenVAS el requerimiento de escaneo. El resultado del escaneo es un reporte en formato NBE (formato históricamente utilizado por Nessus y hoy en día utilizado por OpenVAS). Dicho reporte es procesado por el parser que hemos desarrollado, el cual separa y clasifica cada informe de acuerdo a su severidad (alta, media, baja o ninguna).

El parser

El parser toma como entrada un reporte NBE, lo procesa e inserta los resultados en la base de datos. Para poder realizar este procesamiento hay que comprender la estructura que posee el formato NBE. Este es un archivo de múltiples líneas en texto plano, donde cada una de las líneas representa un resultado. Y en cada uno de ellos tendremos, separados por el carácter especial |(pipe), los siguientes datos:

- Tipo de resultado (el que nos interesa es el tipo 'result')
- Objetivo
- IP
- Puerto
- OID (Identificación del plugin de OpenVAS utilizado)
- Tipo
- Descripción

El parser simplemente toma cada línea, separa los datos y los prepara para ser insertados en la base de datos. Antes de que esto ocurra, hay ciertos datos que tenemos que extraer del campo de descripción de cada resultado, como ser, si posee algún CVE[27] relacionados y, más importante aún, si posee algún factor de riesgo. Este factor es el utilizado para clasificar el nivel de severidad que posee la vulnerabilidad (alta, media, baja o ninguna).

Visualización de reportes

Una vez que se han insertado los reportes en la base de datos, el sistema es el encargado de mostrar al responsable del servidor un reporte del estado actual de las vulnerabilidades detectadas. Pudiéndose observar, para cada una de ellas:

- La dirección IP y el puerto donde corre el servicio afectado
- El plugin de OpenVAS utilizado para detectar la vulnerabilidad
- Una lista de links a CVE
- Una breve explicación de la vulnerabilidad
- La posible solución

Con esta información el responsable del equipo tiene las herramientas para decidir cuales de las vulnerabilidades detectadas pueden representar un problema real y elegir bloquear el tráfico entrante al puerto en que escucha el servicio vulnerable hasta que sea corregido.

Por otra parte, el tener los resultados de varios escaneos en la base de datos nos permite realizar un reporte estadístico histórico de cómo fue evolucionando el estado del servidor en cuestión a lo largo del tiempo. Esto permite a su vez visualizar un reporte en particular de cada escaneo realizado.

Ver figura 5.8.

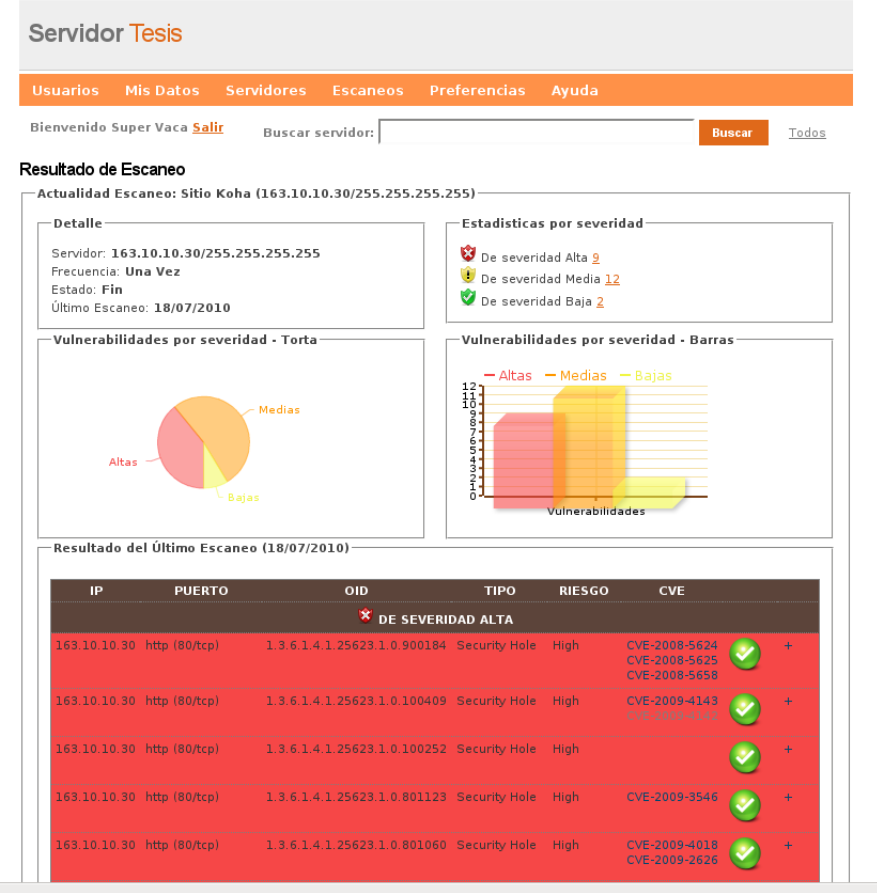

Figura 5.8: Resultado de un escaneo para un servidor

Configuración necesaria

Para empezar a utilizar el servicio es necesario indicar al sistema determinados parámetros que definan con qué servidor de OpenVAS nos vamos a conectar para solicitarle los análisis. Para poder realizar esta conexión nuestra aplicación va a utilizar un cliente OpenVAS, cuya instalación es precondición necesaria para disponer de esta funcionalidad.

Las preferencias del sistema que un usuario con rol administrador debe configurar son:

- openvas-server: IP del servidor de openvas.
- openvas-port: Puerto del servidor de openvas.
- openvas-user: Usuario para conectarnos al servidor de openvas.
- openvas-pass: Password para conectarnos al servidor de openvas.
- openvas-client: Ejecutable del cliente de openvas local.

El servidor de OpenVAS de manera similar a lo que ocurre con el firewall no necesita estar en el mismo equipo que nuestra aplicación.

5.3.5. Alertas

ABM de solicitudes de monitoreo de tráfico

Una vez que un servidor es dado de alta y se ha habilitado el monitoreo de tráfico para dicho equipo, se presentará al dueño del servidor un resumen de las alertas que Snort vaya detectando.

Una vez que Snort generó un alerta será el trabajo de Barnyard registrarla en la base de datos, para que luego el sistema pueda procesarla.

¿Cuando procesamos las alertas?

El procesamiento de alertas se realiza mediante una tarea programada, la cual invoca a un script que es el encargado de procesar las alertas que fueron registradas por el Barnyard2, y de ingresarlas, si es que corresponde, en la lista de eventos que maneja el sistema.

Se registra internamente la última vez que se procesaron las alertas del Snort para trabajar sólo sobre las nuevas y evitar el reproceso innecesario de las que ya fueron parseadas.

¿Cómo vemos las alertas?

El responsable selecciona un servidor y las alertas se mostrarán en una pantalla de resumen clasificadas de distintas maneras. Estos criterios de visualización y clasificación de las alertas son variados, sólo a modo de ejemplo nombramos algunos de los que usamos:

- Por origen (cuando es hacia el servidor)
- Por destino (cuando es desde el servidor)
- Por signature (según clasificación de Snort)
- Por severidad del evento (según clasificación de Snort)
- Por naturaleza del evento

La cantidad de días para los cuales se muestran los eventos es modificable en el momento a través de la interfaz, por defecto se muestran los de los últimos 30 días.

También se proveé un histórico de eventos agrupados por semana y clasificados según severidad. La cantidad de semanas que se muestran también es modificable en el momento pero viene con un valor por defecto de 12 semanas hacia atrás. Esta funcionalidad la consideramos de gran valor, ya que el responsable del sistema pueda observar cómo evoluciona la atención de los atacantes sobre su equipo.

Todos estos datos se visualizan acompañados por bonitos gráficos de torta y/o barra, obteniéndose una representación gráfica de la información provista por el sistema. Estos gráficos se han incorporado a través del uso de las librerías de Open Flash Chart [72] para Perl.

Adicionalmente y mediante la integración con las librerías geoip[73] y la API de google maps[74] se puede visualizar en un mapa mundial el origen o destino de los ataques, según sea el caso.

A su vez también es posible acceder a ver más detalles de los eventos, pudiendo acceder a listados de los eventos según el criterio elegido en el resumen. Desde estas listas podemos inspeccionar los eventos, pudiendo llegar a examinar hasta

encabezados ip, tcp, udp, icmp y los payloads. Las alertas, dependiendo del tipo que sean, puede tener asociado un número de CVE, en cuyo caso se presenta un link con la referencia para el acceso directo a más información sobre el incidente.

Adicionalmente, desde estas listas podemos solicitar el bloqueo del tráfico que cumple con las características del evento, esto se realiza de manera manual, mediante un click en un botón. Una vez que el tráfico fue bloqueado puede ser desbloqueado tanto desde este listado como desde el listado de tráfico relacionado al servidor.

Otra funcionalidad interesante provista por la aplicación es que en el momento de ingresar el sistema brinda un resumen de los eventos que fueron detectados entre la fecha en que ingresamos anteriormente y el inicio de la sesión actual. Ver figura 5.9.

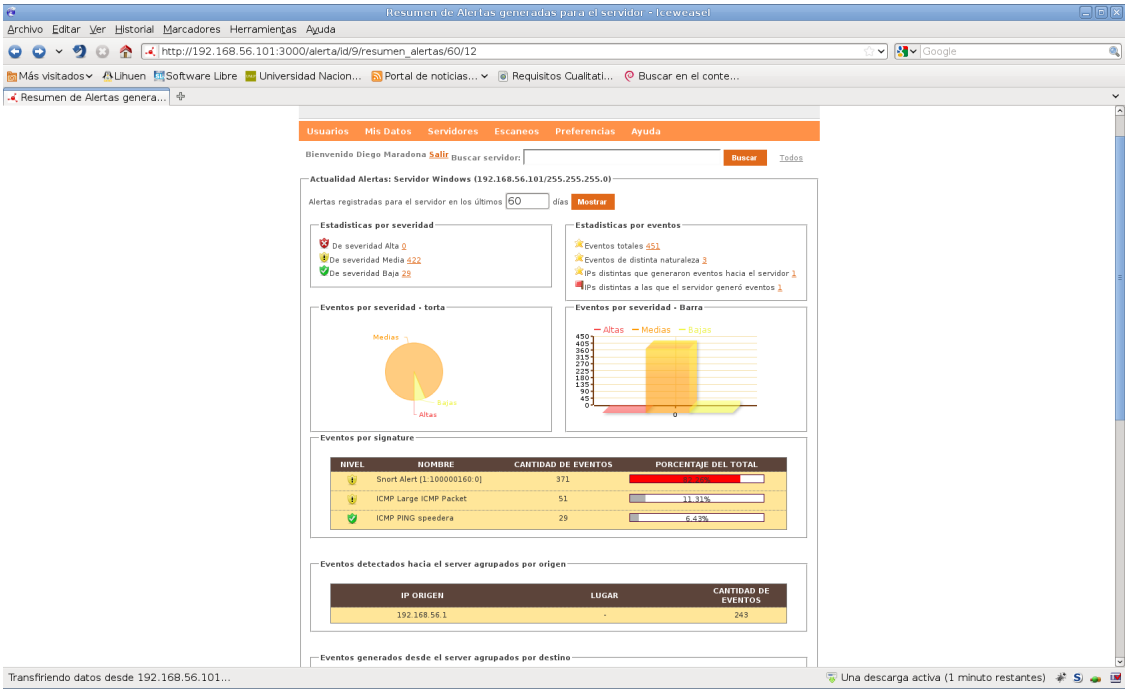

Figura 5.9: Pantalla resumen de alertas

5.3.6. Estándares web

A la hora del desarrollo respetamos los estándares de la W3C[75], tanto para el lenguaje de marcado de hipertexto como para las hojas de estilo utilizadas. Ambos temas son tratados brevemente a continuación, pero podría ampliarse a varias páginas o incluso un par de capítulos más.

¿Qué es XHTML 1.0?

A lo largo del tiempo la W3C fue el origen de varias RFCs que fueron rigiendo o estandarizando el desarrollo web. XHTML 1.0 fue la primer recomendación para XHTML y es una reformulación de HTML 4.01 en XML.

XHTML 1.0 fue el primer cambio mayor que sufrió HTML 4.0 desde que fue realizado en 1997. En definitiva es el resultado del trabajo continuo de la W3C a través de HTML 2.0, HTML 3.2, HTML 4.0 y HTML 4.01.

Fue especificado en 3 variantes, cada una con su propio DTD que lo valida:

- XHTML 1.0 Strict -Se utiliza en combinación con W3C CSS. Siguen al pie de la letra las convenciones marcadas separando completamente las etiquetas de formato del contenido apoyándose exclusivamente en las CSS. Algunos tags como center o font no se permiten más.
- XHTML 1.0 Transitional - Permite la utilización de algunas etiquetas de formato dentro del código de las páginas para ser soportadas por navegadores antiguos que tienen problemas con las CSS. Sirve para facilitar la migración de código desde HTML 4 a XHTML.
- XHTML 1.0 Frameset - Se utiliza cuando se divide la pantalla en varios frames.

¿Qué es CSS 2.1?

CSS es un lenguaje de hoja de estilo que permite agregar elementos de estilo (como setear la fuente o incluir espacios) a documentos estructurados (como HTML y XML). Mediante la separación del estilo y el contenido de los documentos CSS simplifica el mantenimiento y desarrollo de los sitios. Por ejemplo un desarrollador podría estar encargado de generar un XML que represente los datos sin preocuparse realmente como van a mostrarse y otro preocuparse solamente por elegir colores, fuentes, imagenes, sonidos, etc.

La recomendación CSS 2.1 está construida sobre la CSS2, la cual a su vez fue desarrollada a partir de la CSS1. Soporta entre otras cosas hojas de estilo específicas para el medio en que va a navegarse el sitio, por ej para que el autor pueda preparar sus documentos de manera que pueda ser visualizado correctamente tanto para navegadores visuales, impresoras, dispositivos braille, dispositivos móviles,etc.

CSS 2.1 is un derivado y está desarrollado para reemplazar CSS2 y es la última recomendación al momento que escribimos este trabajo. En el futuro aparece CSS3 pero por ahora es solamente un DRAFT.

¿Cuáles respetamos?

La w3c tiene un validador on line[76] que nos permite verificar si nuestro software respeta los estándares o no, como resultado de la evaluación podemos decir que nuestro trabajo respeta XHTML 1 en su rama Strict[77] y CSS en su versión 2.1 [78]. Ver figuras 5.10 y 5.11.

Además se puede apreciar los íconos de validación en cada una de las pantallas de nuestro sistema como pie de página.

Figura 5.10: Validación XHTML 1.0

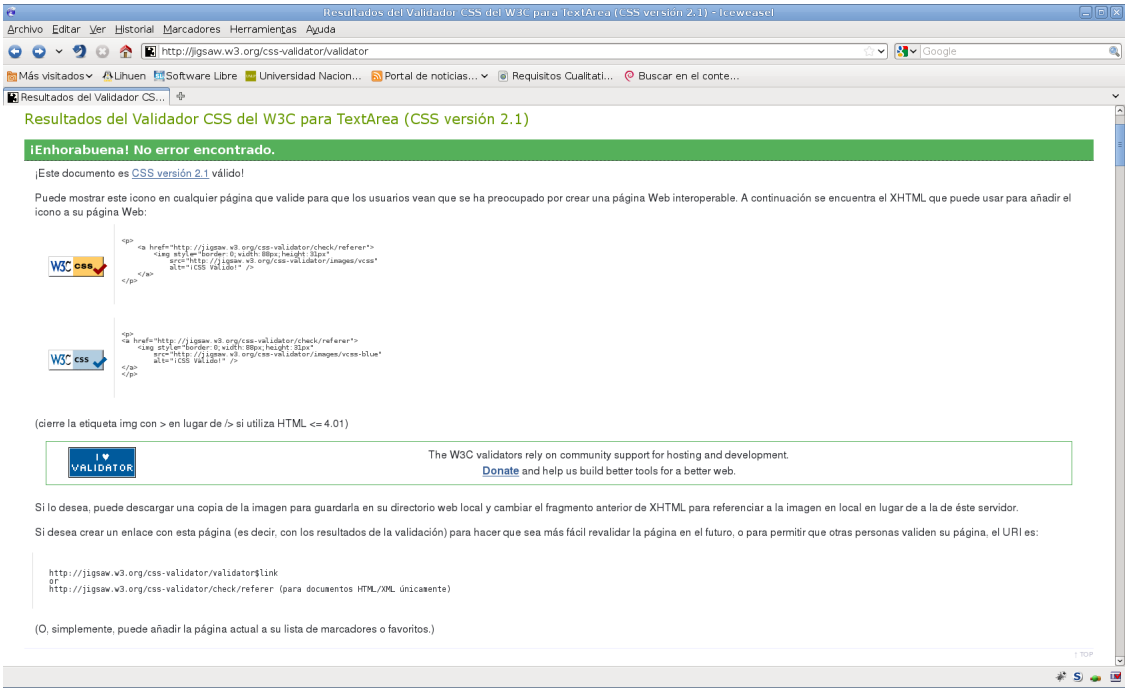

Figura 5.11: Validación de CSS 2.1

5.3.7. Otros aspectos que vale la pena mencionar

El menú preferencias del sistema nos provee un número de variables que utilizamos para regular el funcionamiento y permiten la flexibilización de la instalación e integración de las distintas componentes de nuestra aplicación. Ver figura 5.12.

Se incorporó un lector de noticias o RSS[4] para los usuarios, de manera tal que

[4]RSS (RDF Site Summary or Rich Site Summary): es un formato XML para compartir contenido en la web. Los usuarios que se suscriben a fuentes de contenidos los reciben directamente con un lector que interpreta el XML y se los presenta.

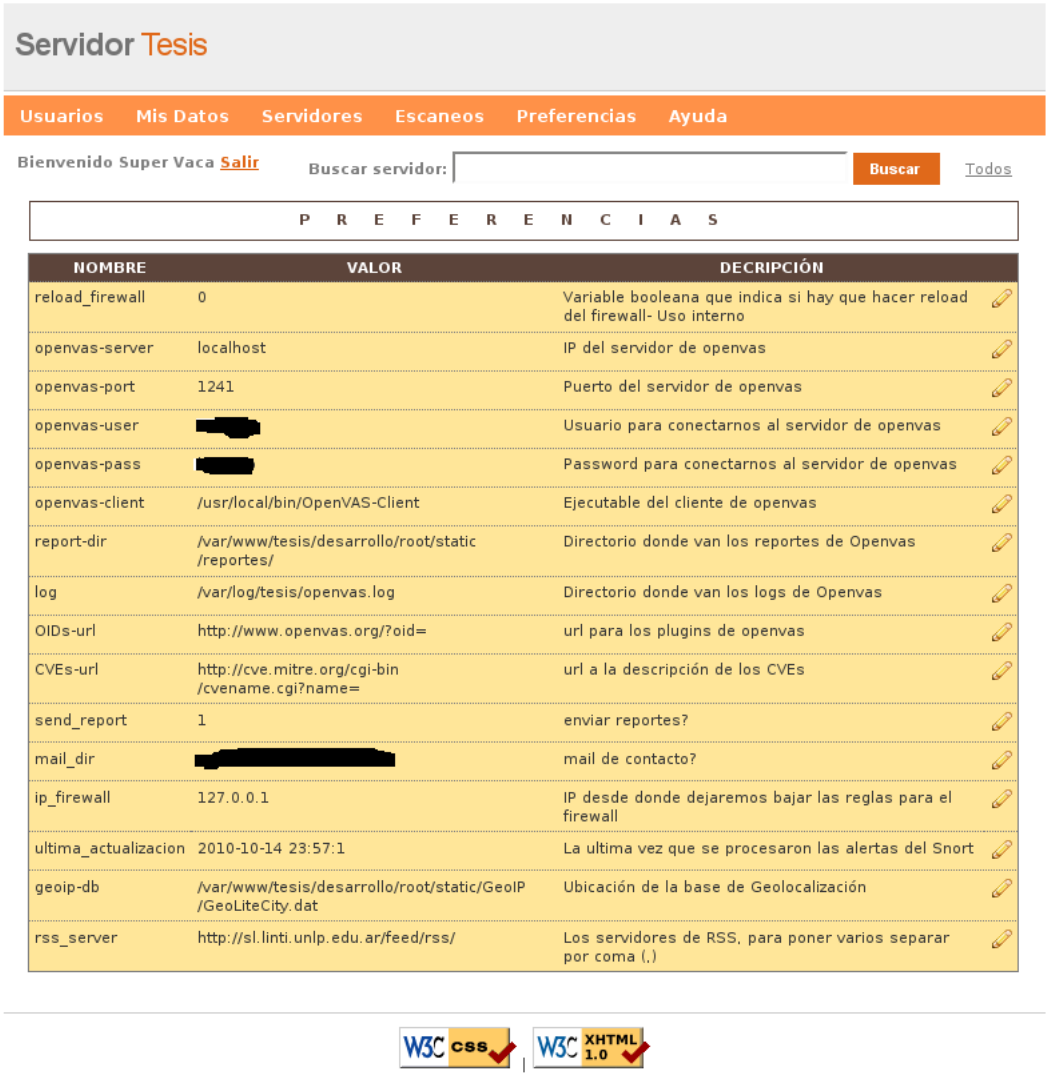

Figura 5.12: Preferencias del Sistema

cuando inicia sesión en la aplicación el usuario accede directamente a un listado de noticias, pudiendo por ejemplo informarse rápidamente sobre las novedades de seguridad que publique alguno de los sitios que se especializan en el tema. Los sitios fuente de estas noticias son configurables desde las preferencias del sistema y pueden ser múltiples.

Como funcionalidad adicional, en el momento que se inicia sesión en el sistema, se provee al usuario un resumen de las últimas acciones que se realizaron con sus credenciales, como ser, último login, último logout, intentos fallidos de inicio de sesión e información sobre la sesión actual. Ver figura 5.13.

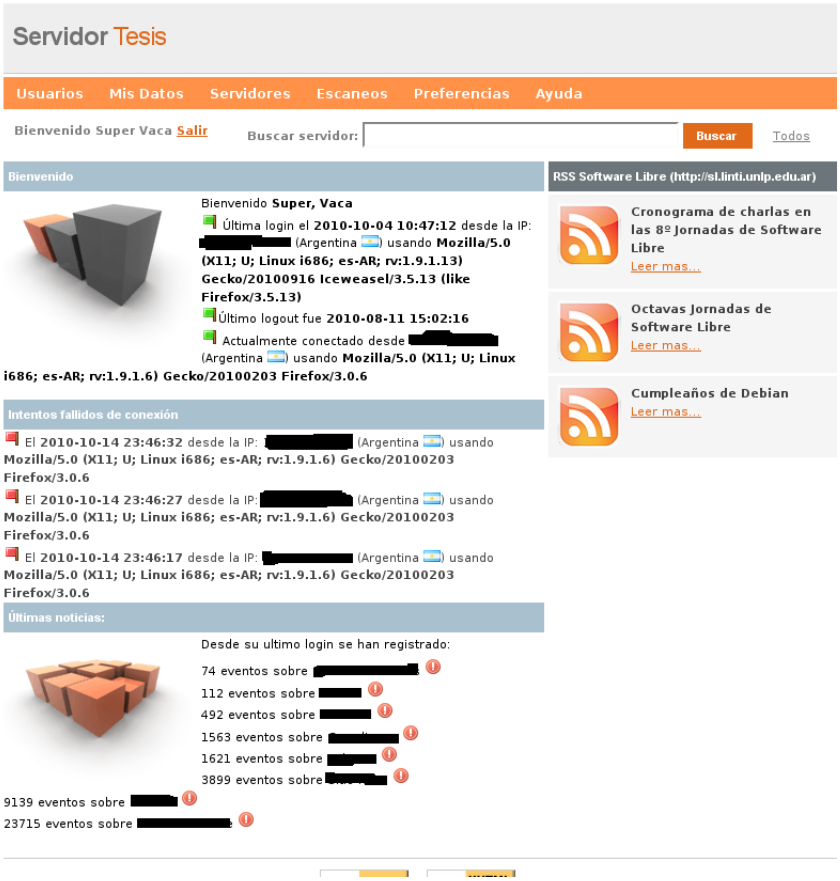

Figura 5.13: Una vez que iniciamos sesión

5.4. Radiografía del sistema

5.4.1. El código

Como ya mencionamos lo que empezó como un prototipo terminó siendo un desarrollo de software con cierta complejidad íntegramente desarrollado en Perl utilizando el Framework Catalyst, en esta sección trataremos de dar una idea detalles del trabajo que involucró el desarrollo.

Como mencionamos se respetó el modelo MVC, utilizamos Template Toolkit y FormFu para el View, DBIx para el Model y Catalyst Controller para el Controller.

Para mencionar algunas de las librerías en las que nos apoyamos para nuestro desarrollo podemos enumerar:

- Catalyst::Controller[79]
- Catalyst::Controller::HTML::FormFu[80]
- Chart::OFC2 y subclases (Axis,Bar,Line,Pie)[81]
- DBIx::Class[82]
- DateTime
- Jquery[83]
- Json[84]
- Google jsapi[74]

Al ser un desarrollo distribuido entre dos personas, tuvimos que utilizar un mecanismo que nos permitiera mantenernos sincronizados, ya que ir combinando el desarrollo de forma manual es muy complicado, termina siendo un desperdicio de tiempo, y generalmente una fuente de generación de errores.

El mecanismo elegido fue utilizar un sistema de control de versiones, concretamente optamos por Subversion[85], uno de los sistemas de versionado bajo licencia de software libre más difundidos actualmente, muy aceptado y que cuenta con clientes para múltiples plataformas.

Siendo mas concretos podemos mencionar que después de más de 340 commits, nuestro proyecto involucra aproximadamente 100 archivos distintos desarrollados por nosotros. Desglosándolo tenemos 50 archivos diferentes para el View (1369 líneas de código), 17 para el Controller (289 líneas de código) y 33 para el Model (1089 líneas de código). Es decir que tenemos casi 2800 líneas de código, lo cual si bien no es indicativo que se pueda tomar para evaluar la calidad del trabajo, sirve a la hora de cuantificar el trabajo realizado.

5.4.2. El diagrama de clases

Como mencionamos previamente, utilizamos programación orientada a objetos en el desarrollo. La idea de esta subsección era poder presentar el diagrama de clases, pero cuando lo generamos nos dimos cuenta que, dada la magnitud del diagrama, nuestra idea carece de sentido ya que para poder explicarlo tendríamos que extendernos muchas páginas más. Nuestro diagrama involucra 57 clases definidas por nosotros y 25 clases adicionales entre las del framework y las auxiliares.

Incluso para poder presentar el gráfico del esquema a modo simplemente de referencia tuvimos que dividirlo en 2, por un lado la parte del View y el Controller (Ver figura 5.14) y por el otro las clases del Model (Ver figura 5.15), y aún así continúa siendo ilegible.

5.4.3. El modelo de la base de datos

Como dijimos la base de datos del sistema está definida sobre un motor MySQL[63] y cuenta con 39 tablas que manejan los distintos datos, en este esquema de tablas esta tanto las que necesita el Barnyard para guardar lo que detecta Snort y que luego nosotros usamos como las tablas que directamente definimos nosotros.

Entender el esquema de base de datos de Snort fue muy complicado ya que no encontramos documentación que nos ayudara sino que hubo que hacer ingeniería inversa y a fuerza de prueba y error descubrimos la información necesaria.

Ver figuras 5.16 y 5.17.

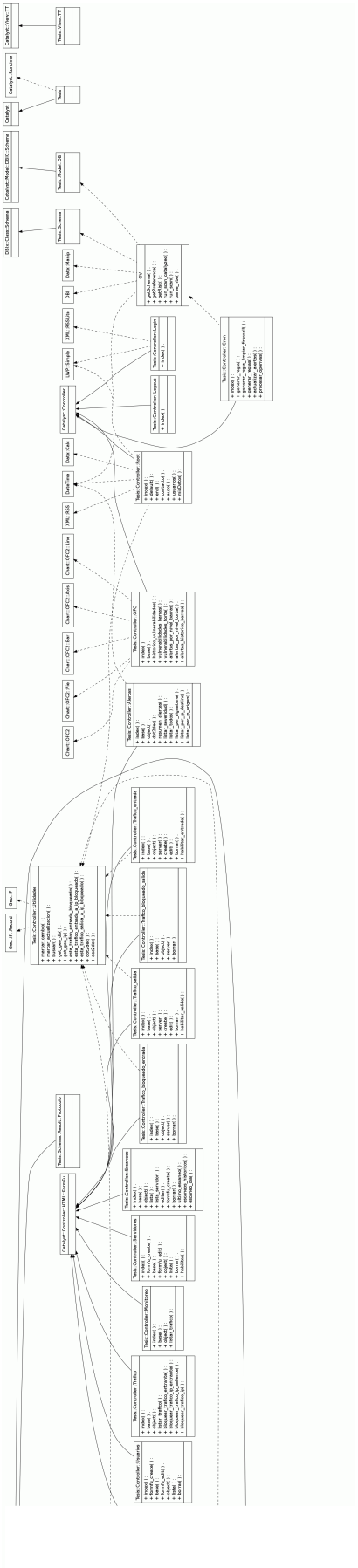

Figura 5.14: Diagrama de clases parte 1 - View-Controller

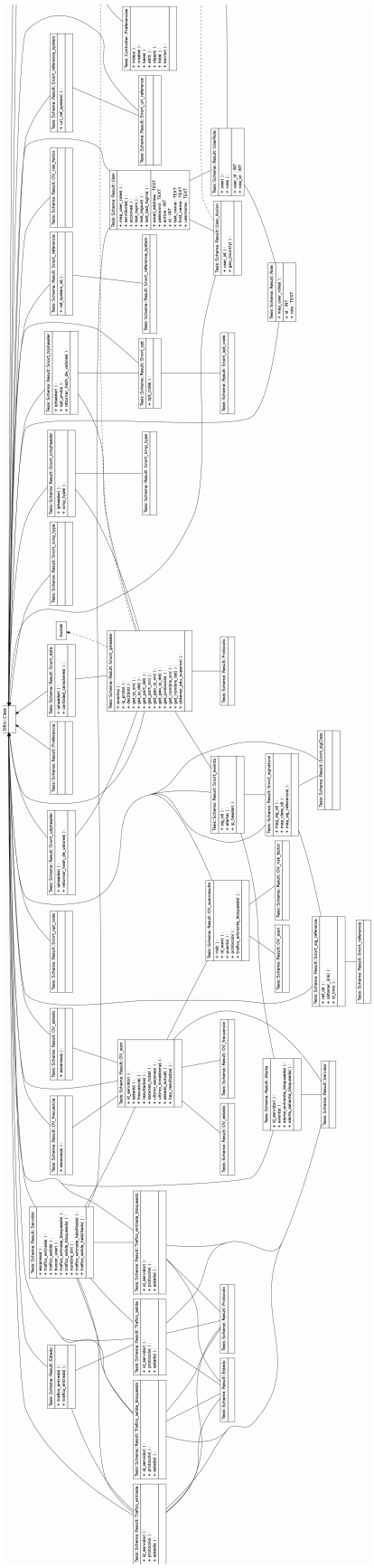

Figura 5.15: Diagrama de clases parte 2 - Model

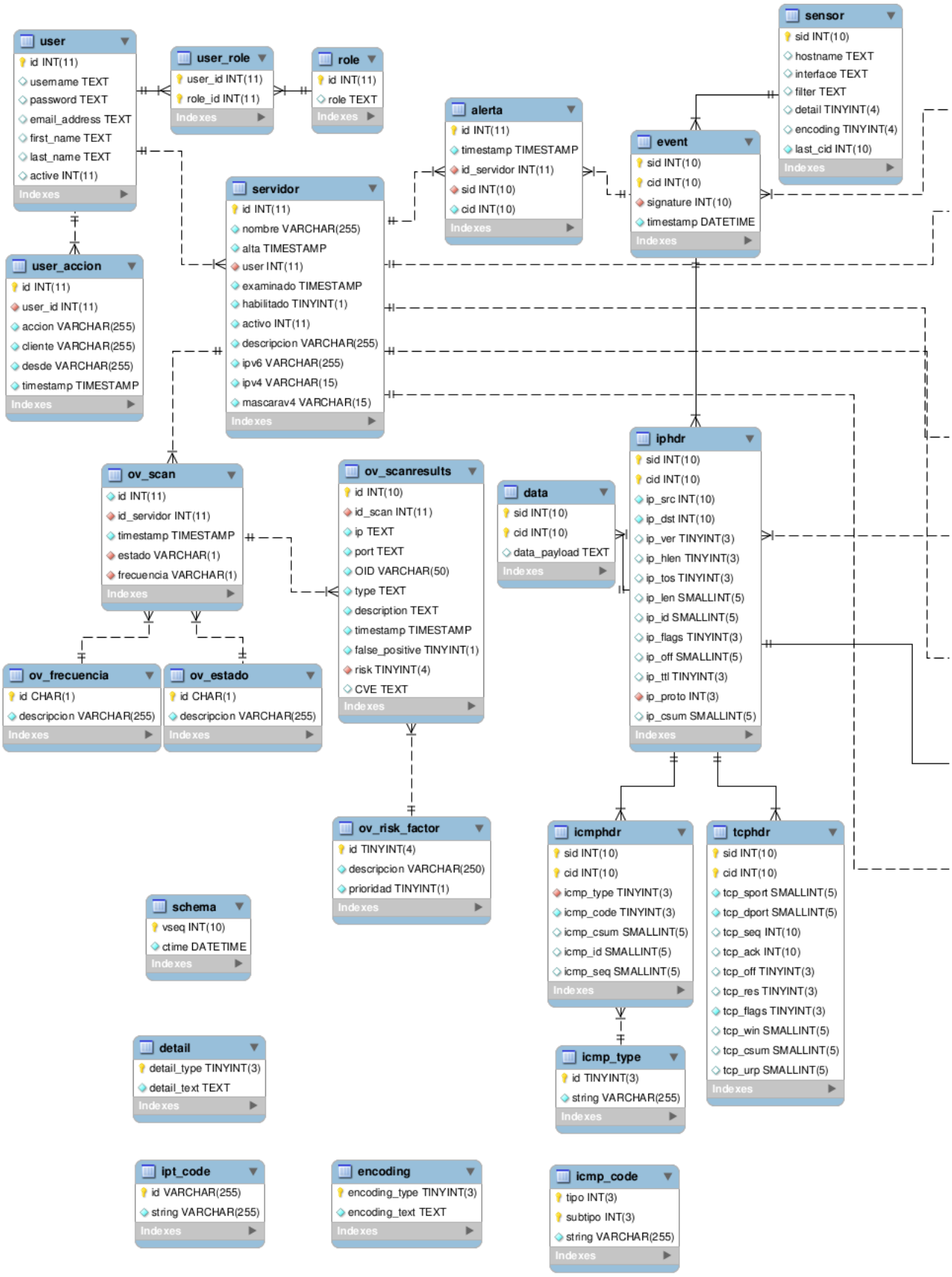

Figura 5.16: Modelo de la base de datos parte 1

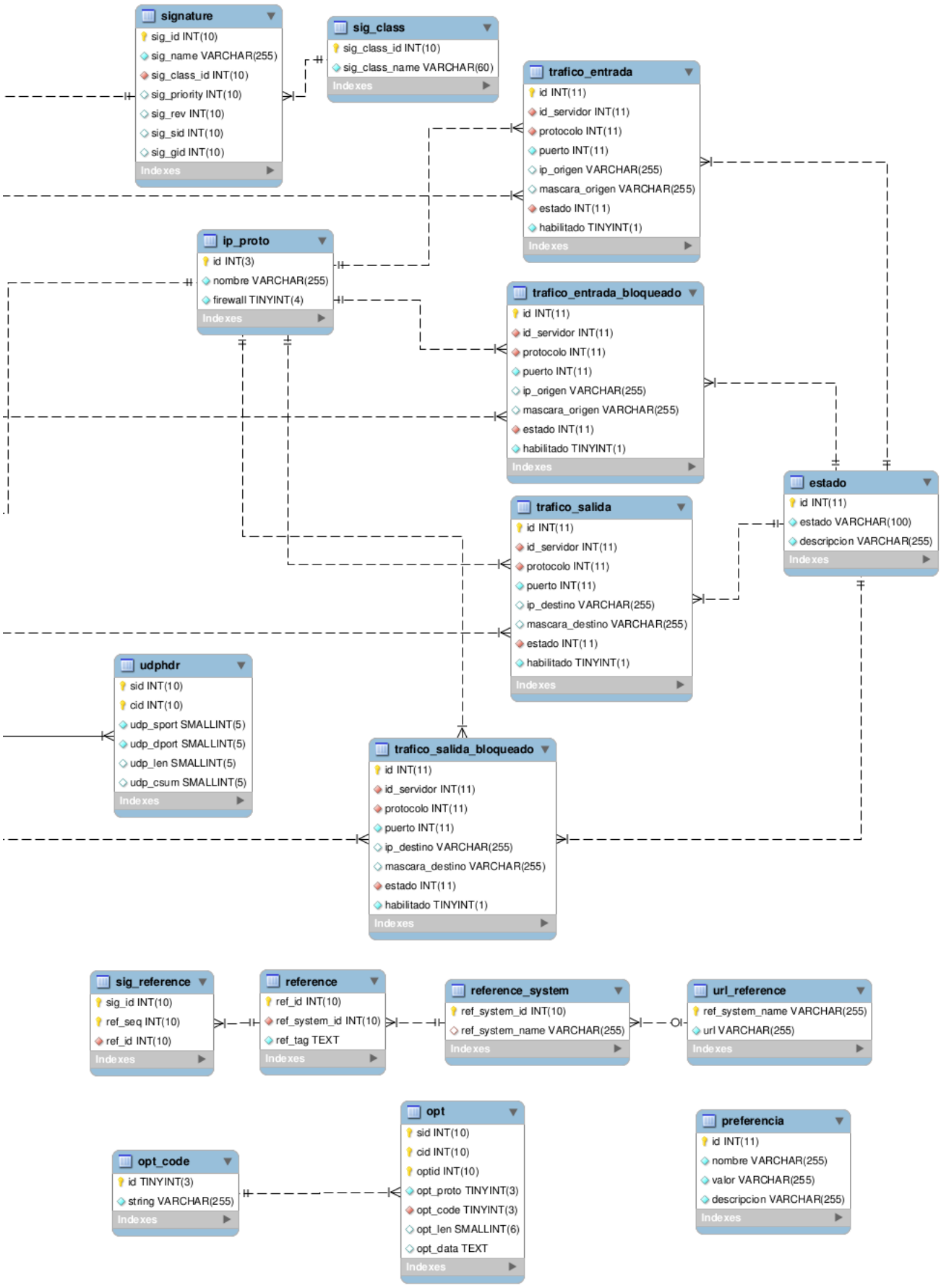

Figura 5.17: Modelo de la base de datos parte 2

Capítulo 6

Conclusiones y temas de trabajo futuro

6.1. Conclusiones

La verdad que el título del capítulo asusta cuando empezamos a escribirlo, ya que son muchas las ramas que podemos recorrer para completar esta parte del trabajo. De todas maneras vamos a tratar de resumir los resultados obtenidos.

Por un lado este documento escrito como trabajo de tesina termina siendo un texto que puede utilizarse como texto de referencia para cualquiera que quiera iniciarse en varios temas como son Firewalls, escaners de puertos o de vulnerabilidades e IDS.

Adicionalmente a resumir los conceptos, brinda un análisis de las aplicaciones de software libre existentes en el mercado hacia fines del año 2009 que implementan estos conceptos y la justificación de los productos seleccionados.

Gracias al hecho de trabajar en equipo de manera distribuida y en paralelo tanto para el desarrollo como para escribir este documento nos vimos en la necesidad de buscar algún entorno colaborativo. Para la escritura del texto encontrando en Subversion y Latex las herramientas ideales. Como los archivos de latex no son binarios sino que son simples archivos de texto plano sin cosas raras a partir de subversion pudimos ir combinando lo que cada uno escribió y mantener el versionado del trabajo sin inconvenientes. Adicionalmente al usar Latex nos olvidamos de los dolores de cabeza que siempre uno encuentra con los formatos, tipos de letras, tabulaciones, indices, inserción de figuras y muchos etcéteras que siempre aparecen en el uso de las distintas suites de ofimática tradicionales, tanto las libres como las de software privativo. Si bien subversion ya lo conocíamos, tuvimos que aprender Latex para poder escribir este informe, lo cual también consideramos un hallazgo y un aporte excelente para nuestro desarrollo como profesionales. Como primer conclusión podemos decir que aprendimos a usar y a querer a Latex.

En segundo término queremos recalcar que lo que empezó en la propuesta de tesis como un "prototipo" terminó siendo un sistema completo que tiene aun más funcionalidad de la que imaginamos que tendría el sistema final que ibamos "a prototipar". Es más, tan convencidos estamos de que el desarrollo es un producto completo que puede ser de utilidad que pensamos ponerlo en producción en la DMZ de nuestro de lugar de trabajo próximamente, aunque ya nos referiremos a estas intenciones en la sección de trabajo a futuro.

Con lo anteriormente expuesto podemos decir que el resultado final es un complejo desarrolló que permite la administración descentralizada de una red tipo DMZ

mediante un software que integra sin problemas herramientas de distinta naturaleza y que tienen diferentes complejidades de instalación y configuración como son Iptables, Snort y Openvas. En el aplicativo cada servidor registrado tiene un dueño que es el que puede requerir permisos de tráfico, solicitar análisis de vulnerabilidades y revisar las alertas que registra un NIDS relacionadas con su equipo.

Todo ésto gestionado a través de una simple interfaz web que respeta los estándares más actuales de la W3C al día de hoy XHTML 1.0 Strict y CSS 2.1. A través de ella la gestión puede ser realizada por cualquier usuario, sin que necesariamente deba ser un experto, el único requisito es poseer un conocimiento mínimo de los parámetros de configuración de su servidor, por ejemplo ip del servidor y puertos que debe tener habilitados.

Entonces, podemos decir que el software cumple con los estándares de desarrollo web, respeta las licencias y está desarrollado respetando los lineamientos del Catalyst Framework. Estos factores hacen que estemos orgullosos del producto final ya que consideramos que está a la altura de cualquier aplicación que se pueda conseguir en el mundo del software libre.

En cuanto al código resultante podemos decir que agregar o modificar funcionalidad es relativamente sencillo para cualquiera que entienda Catalyst. Y que si bien la curva de aprendizaje del framework fue bastante extensa, una vez que la superamos terminamos aprovechando y disfrutando de las bondades que nos proporciona. Por ello podemos decir que dimos nuestros primeros pasos en el uso de Catalyst.

Como idea final queremos expresar que si bien hoy en día hay muchos detractores de la seguridad a nivel de infraestructura en favor de la seguridad a nivel aplicación, cuya fundamentación se basa en que lo que tienen como objetivo actual los atacantes son las aplicaciones que se hostean, nosotros creemos que nuestra arquitectura al ser un híbrido entre seguridad a nivel de aplicación y seguridad a nivel de infraestructura que favorece ambos aspectos, ya que facilita tanto la administración como el seguimiento. Por otro lado pensamos que la seguridad a nivel de infraestructura nunca debe dejarse de lado ya que además de ser indispensable de por sí, si está bien aplicada colabora en reducir el impacto de los problemas que puede ocasionar por ejemplo una aplicación web que sea vulnerada.

6.2. Trabajos Futuros

A medida que avanzamos en el desarrollo el abanico de posibilidades a futuro creció y creció, a continuación suministraremos una lista de los que nos parecen los pasos más razonables a seguir:

- Como primer paso nos vemos tentados a retribuir a la comunidad del software libre, ya que en gran parte este trabajo pudo desarrollarse gracias a ella. La idea es registrar el proyecto en sourceforge y publicar el software bajo alguna licencia libre para que otros lo puedan aprovechar.

- Instalar el software en un ambiente producción para poder, por un lado disfrutar de sus bondades y por otro lado realizar los ajustes y mejoras que surjan, que seguramente van a ser más evidentes ante un uso intensivo.

- Mejorar la documentación para el usuario final y el desarrollador.

- Empaquetar la aplicación para facilitar su distribución, por ejemplo, para Debian GNU/Linux.

- Desarrollar una guía de instalación.
- Agregar la posibilidad de reglas por defecto para el Firewall: independientemente de lo definido para cada servidor que exista un conjunto de reglas definidas previamente.
- Mejorar la parte de auditoría del sistema, hoy no existe una bitácora de las acciones de cada usuario con sesión en el sistema.
- Agregar la posibilidad de exportar tanto las alertas del Snort como los resultados de escaneos a varios formatos (planillas de cálculo, pdf y, en el caso de los escaneos, en formato NBE).
- Permitir subir reportes de escaneos antiguos con fines estadísticos.
- Desarrollar un módulo de reportes vía mail, que envíe un resumen tanto de las alertas detectadas como de las vulnerabilidades encontradas.
- Agregar el soporte para IPV6.
- Agregar la posibilidad de utilizar múltiples IDS, hoy funciona con uno solo.

Bibliografía

[1] Daniel Barrett. *Linux security cookbook*. O'Reilly, Beijing ; Sebastopol CA, 1st ed. edition, 2003.

[2] Intrusion detection system - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Intrusion_detection_system.

[3] About the GNU project - GNU project - free software foundation (FSF). http://www.gnu.org/gnu/thegnuproject.html.

[4] Linux online. http://www.linux.org/.

[5] All about linux. http://linuxhelp.blogspot.com/.

[6] N. Freed. Behavior of and Requirements for Internet Firewalls. RFC 2979 (Informational), October 2000.

[7] Modelo OSI - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Modelo_OSI.

[8] T.J. Socolofsky and C.J. Kale. TCP/IP tutorial. RFC 1180 (Informational), January 1991.

[9] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[10] Dirección MAC - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Direcci%C3%B3n_MAC.

[11] James Kurose. *Redes de computadores : un enfoque descendente basado en Internet*. Pearson Addison Wesley, Madrid, 2a. ed. edition, 2003.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[13] Vulnerability scanner - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Vulnerability_scanner.

[14] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[15] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[16] IANA — internet assigned numbers authority. http://www.iana.org/.

[17] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFCs 950, 4884.

[18] K. Sollins. The TFTP Protocol (Revision 2). RFC 1350 (Standard), July 1992. Updated by RFCs 1782, 1783, 1784, 1785, 2347, 2348, 2349.

[19] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), October 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[20] OpenSSH. http://www.openssh.com/.

[21] The apache software foundation. http://www.apache.org/.

[22] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[23] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336.

[24] J. Postel and J.K. Reynolds. Telnet Protocol Specification. RFC 854 (Standard), May 1983. Updated by RFC 5198.

[25] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Standard), May 1996. Updated by RFCs 1957, 2449.

[26] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738.

[27] CVE - common vulnerabilities and exposures (CVE). http://cve.mitre.org/.

[28] SecurityFocus. http://www.securityfocus.com/.

[29] Licenses - GNU project - free software foundation. http://www.gnu.org/licenses/licenses.html.

[30] Free software foundation — working together for free software. http://www.fsf.org/.

[31] The free software definition - GNU project - free software foundation (FSF). http://www.gnu.org/philosophy/free-sw.html.

[32] What is copyleft? - GNU project - free software foundation (FSF). http://www.gnu.org/copyleft/.

[33] Categories of free and nonfree software - GNU project - free software foundation (FSF). http://www.gnu.org/philosophy/categories.html.

[34] The open source definition | open source initiative. http://www.opensource.org/docs/osd.

[35] The GNU general public license - GNU project - free software foundation (FSF). http://www.gnu.org/licenses/gpl.html.

[36] The GNU operating system. http://www.gnu.org/.

[37] Linux and GNU - GNU project - free software foundation (FSF). http://www.gnu.org/gnu/linux-and-gnu.html.

[38] Android - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Android.

[39] Sitio oficial de lihuen. http://lihuen.info.unlp.edu.ar/.

[40] Koha UNLP. http://koha.unlp.edu.ar/.

[41] Daemon news - best software review. http://www.daemonnews.org/200104/bsd_family.html.

[42] www.bsd.org. http://www.bsd.org/.

[43] Berkeley software distribution - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Berkeley_Software_Distribution.

[44] Comparison of BSD operating systems - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Comparison_of_BSD_operating_systems.

[45] GNU/Linux - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/GNU/Linux.

[46] DistroWatch.com: put the fun back into computing. use linux, BSD. http://distrowatch.com/.

[47] Debian – el sistema operativo universal. http://www.debian.org/.

[48] PF: the OpenBSD packet filter. http://www.openbsd.org/faq/pf/.

[49] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[50] netfilter/iptables project homepage - the netfilter.org project. http://www.netfilter.org/.

[51] Tenable network security. http://www.nessus.org/nessus/.

[52] Jay Beale. *Nessus network auditing*. Syngress,, Rockland, MA :, 2004.

[53] Nmap - free security scanner for network exploration & security audits. http://nmap.org/.

[54] Gordon Lyon. *Nmap network scanning : official Nmap project guide to network discovery and security scanning*. Insecure.Com LLC, Sunnyvale CA, 2008.

[55] The NASL2 reference manual. http://michel.arboi.free.fr/nasl2ref/.

[56] OpenVAS - OpenVAS - open vulnerability assessment system community site. http://www.openvas.org/.

[57] Bro intrusion detection system - bro overview. http://bro-ids.org/.

[58] Index of /bro-workshop-2007/slides. http://www.bro-ids.org/bro-workshop-2007/slides/.

[59] Bro Hands-On workshop 2009, the 2nd. http://www.bro-ids.org/bro-workshop-2009-2/index.html.

[60] Snort :: Docs. http://www.snort.org/docs.

[61] InformIT: dissecting snort > feeding snort packets with libpcap. http://www.informit.com/articles/article.aspx?p=101148&seqNum=1.

[62] securixlive.com :: barnyard2. http://www.securixlive.com/barnyard2/index.php.

[63] MySQL :: The world's most popular open source database. http://www.mysql.com/.

[64] The perl programming language - www.perl.org. http://www.perl.org/.

[65] Modelo vista controlador - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Modelo_Vista_Controlador.

[66] Don't repeat yourself - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Don%27t_repeat_yourself.

[67] Ruby on rails. http://rubyonrails.org/.

[68] Django | the web framework for perfectionists with deadlines. http://www.djangoproject.com/.

[69] symfony | web PHP framework. http://www.symfony-project.org/.

[70] CakePHP: the rapid development php framework. pages. http://cakephp.org/.

[71] Catalyst - web framework. http://www.catalystframework.org/.

[72] Open flash chart - home. http://teethgrinder.co.uk/open-flash-chart-2/.

[73] MaxMind - GeoIP perl API. http://www.maxmind.com/app/perl.

[74] API de google maps - google code. http://code.google.com/intl/es-AR/apis/maps/index.html.

[75] World wide web consortium (W3C). http://www.w3.org/.

[76] The W3C markup validation service. http://validator.w3.org/.

[77] W3C XHTML2 working group home page. http://www.w3.org/MarkUp/.

[78] Cascading style sheets level 2 revision 1 CSS 2.1 specification. http://www.w3.org/TR/CSS2/.

[79] Catalyst::Controller - search.cpan.org. http://search.cpan.org/~bobtfish/Catalyst-Runtime-5.80027/lib/Catalyst/Controller.pm.

[80] Catalyst::Controller::HTML::FormFu - search.cpan.org. http://search.cpan.org/~cfranks/Catalyst-Controller-HTML-FormFu-0.06001/lib/Catalyst/Controller/HTML/FormFu.pm.

[81] Chart::OFC2 - search.cpan.org. http://search.cpan.org/dist/Chart-OFC2/lib/Chart/OFC2.pm.

[82] DBIx::Class - search.cpan.org. http://search.cpan.org/dist/DBIx-Class/lib/DBIx/Class.pm.

[83] jQuery: the write less, do more, JavaScript library. http://jquery.com/.

[84] JSON. http://www.json.org/.

[85] Apache subversion. http://subversion.apache.org/.