UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

TRABAJO ESPECIAL DE GRADO
SISTEMA DE AUTENTICACIÓN SIGILOSA CON TERMINAL DE ADMINISTRACION

Br. Charles M. Romestant F.
C.I.:14.486.880

TUTOR
Lic. Walter J. Hernández B.

Introducción

El campo de la seguridad en redes es cada día más exigente, el nivel de interconexión es cada día mayor y la reducción de costos que brinda una estructura de servicios automatizada resulta ser muy atractiva para las corporaciones. La popularidad de la integración de servicios, la banca virtual, y las tienda-E, entre otras, aumenta a cada segundo. Los usuarios, sin saberlo, están usando servicios basados en un sistema que, debido a su diseño original, fue concebido para la interconexión libre y, por lo tanto, no es seguro. La investigación en seguridad en redes tiene como tarea proteger estas comunicaciones, para brindar así seguridad a un sistema inseguro.

Detrás de los servicios que se brindan siempre se encuentra un servidor. Existen muchos tipos de servidores clasificados comúnmente por el tipo de servicio que brindan, a grosso modo, podemos decir que existen los servidores Web, servidores de Bases de Datos, servidores de Archivos, etc. Todos estos servidores tienen algo en común: están expuestos a ataques. Esa exposición es totalmente normal ya que, si no estuviesen expuestos, no podrían tener acceso a ellos las personas que requieren consumir los servicios brindados por cada servidor.

El propósito de este trabajo es implementar un sistema de autenticación sigilosa basándose en la premisa de seguridad en profundidad (también conocida como seguridad en capas). Dicha solución se basará en agregar una capa de autenticación sigilosa a los esquemas existentes, ya que de esta manera se puede reducir drásticamente los riesgos a los cuales están expuestos los servidores.

El trabajo se basa extensamente en las tesis de pregrado y maestría de Sebastien Jeanquier (Royal Holloway university of London) y Reinderd Gordon Nathan deGraaf (University of Calgary), debido al estudio meticuloso que ambos hicieron sobre las técnicas de autenticación sigilosa.

El alcance de este trabajo es el de las redes que trabajan sobre la pila de protocolos comúnmente conocida como TCP/IP, ya que el estudio de las

debilidades, fuerzas, técnicas de ataque y defensa conocidas está directamente ligado a esta pila de protocolos.

El trabajo está dividido en cinco capítulos los cuales serán brevemente descritos a continuación:

El primer capítulo presenta el marco teórico para este trabajo especial de grado. Se presentan a grandes rasgos las bases de criptografía, tecnologías de redes, seguridad en capas y autenticación sigilosa, las cuales son fundamentales para el propósito de este trabajo.

El segundo capítulo presenta el diseño de una propuesta. Primero se describe el problema que se tiene, luego se adentra en el diseño de una idea de la cuál se deriva una solución. Seguidamente se describen los componentes de la solución y los detalles del protocolo que será utilizado en dicha solución.

El tercer capítulo consiste en los detalles de la implementación. Se presentan detalles de la solución , algunas configuraciones y los distintos componentes.

El cuarto capítulo presenta las pruebas y los resultados de estas pruebas. Se prueban todos los componentes en toda variedad de configuraciones para mostrar las funciones y el rendimiento de la solución.

El último capítulo presenta las conclusiones y recomendaciones realizadas por el autor de este trabajo.

Marco Teórico:

Protocolos de red:

Internet fue diseñado en los años 60, 70 y 80 como un sistema de comunicaciones robusto entre diversas redes locales. [COMER96][JEAN06] Su diseño se basa en una pila (comúnmente llamada “stack” por su nombre en inglés) de cuatro capas de protocolos:

- Capa de aplicación: responsable de codificar y descodificar los paquetes de manera que lo entiendan las aplicaciones. También provee los servicios que sean necesarios y que hayan sido provistos en las capas inferiores.
- Capa de transporte: responsable de la comunicación inter-proceso y de conexiones confiables.
- Capa de Internet: responsable del direccionamiento global, y enrutamiento entre segmentos físicos de red.
- Capa de acceso a la red: responsable de la comunicación entre nodos.

Cada capa brinda servicio a la capa inferior. [COMER96][ERIC03] Cuando se desea enviar una información, ésta pasa por las distintas capas (hacia abajo) las cuales la van transformando, hasta que llega a la capa física que se encarga de codificar la información en el medio. Cuando la información se recibe, cada capa deshace los cambios realizados y la envía a la capa superior, hasta llegar a la capa de más alto nivel en donde es interpretada.

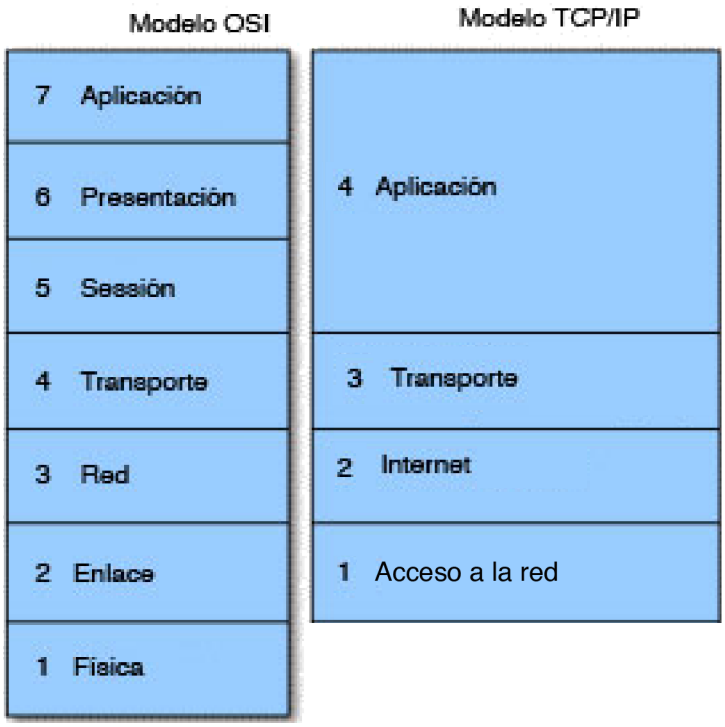

Fig. ## Modelo de Referencia OSI / Stack TCP/IP (reproducción).

La infraestructura de Internet se basa en una serie de enrutadores (routers) interconectados, cuya única función es encaminar la información enviada desde su origen hasta el destino.

Protocolos TCP y UDP:

Los protocolos TCP y UDP (de sus siglas en inglés Transmition Control Protocol y User Datagram Protocol) no son los únicos protocolos de la pila, pero son los que nos interesan en este trabajo.

TCP es un protocolo orientado a conexión (statefull) que le permite a dos hosts crear una conexión entre los dos e intercambiar información. Una conexión en este sentido se puede definir como un acuerdo de comunicación con ciertas reglas establecidas. [COMER96] Esta conexión se establece mediante la

negociación en tres pasos o “Three way handshake” (ver Fig. ##). La mayoría de las aplicaciones , como los chats o los e-mails, usan conexiones TCP.

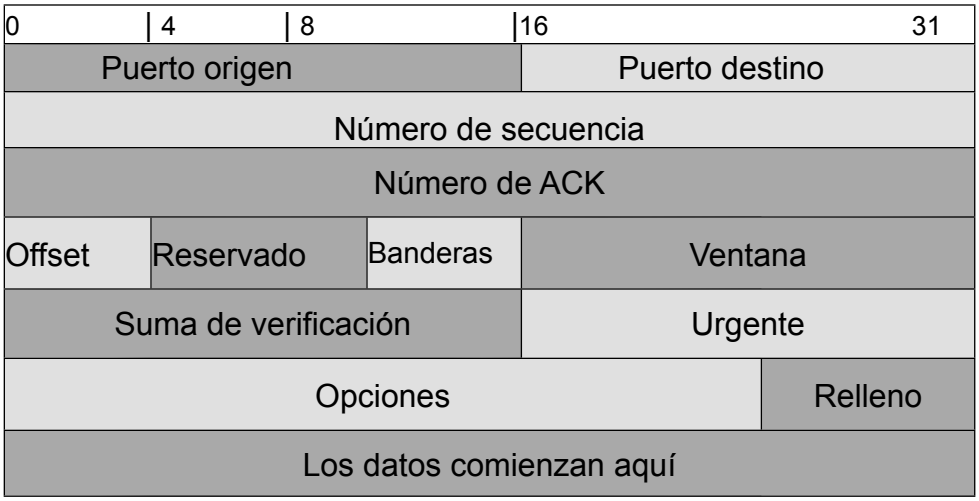

Fig. ### Formato trama TCP. [COMER96]

UDP es un protocolo no orientado a conexión (stateless) [COMER96]. No se crea una conexión formal entre hosts, si un host desea comunicarse con otro host, simplemente lo hace y ya. No recibirá confirmación de recepción. La ventaja de UDP sobre TCP es la velocidad, lo que lo hace mas adecuado para aplicaciones que requieren transmisión rápida de paquetes, sin importar mucho que lleguen todos los paquetes (audio en vivo, VoIP, etc.).

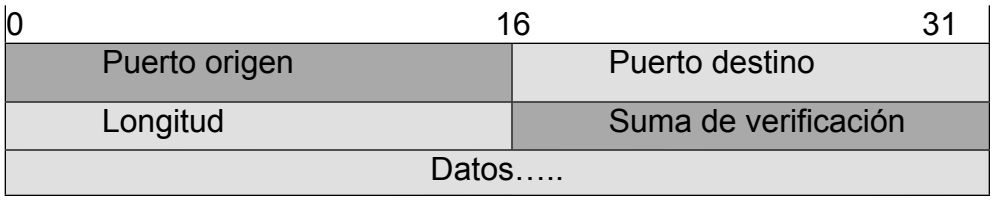

Fig. ### Formato datagrama UDP. [COMER96]

Estos dos protocolos son la base de muchas de las comunicaciones en Internet.

Inicio de una conexión TCP: Three way handshake.

El proceso de negociación en 3 pasos o “Three way handshake” es el protocolo que utilizan dos hosts para establecer una conexión TCP entre sí. Es un acuerdo preestablecido de “modales” sin el cual la creación de una conexión entre dos hosts no sería posible.

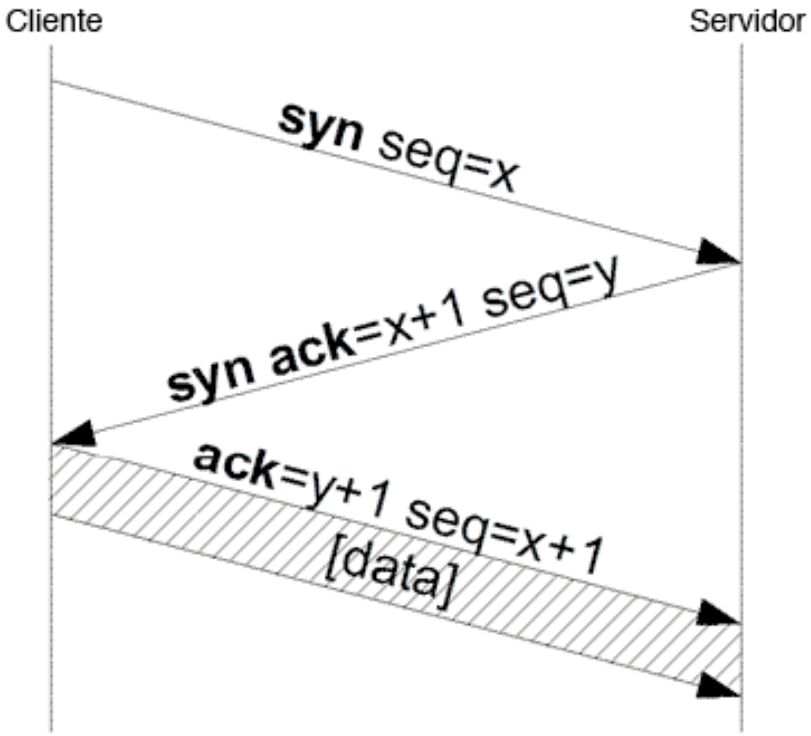

Fig. 2.3 Three way handshake. [JEAN06] (reproducción)

Este proceso requiere que el firewall esté configurado para que permita el paso de los mensajes a los distintos hosts. Si el firewall no está configurado para permitir el paso de estos mensajes, entonces la conexión TCP nunca se terminará de crear.

Puertos:

Una definición simplificada de un puerto seria una puerta virtual a un computador, que ayuda a identificar cual trozo de información recibida va para cual aplicación.

Un puerto es un entero de 16 bits que representa una abstracción lógica sobre la interfaz de red. Estos puertos son representados por un entero sin signo de 16 bits, lo que nos da una cantidad posible de 65536 puertos.

Cuando el número de puerto es usado, se incluye en los paquetes de capa de red, como puerto origen, y puerto destino. Estos números de puertos no sólo son leídos e interpretados por el emisor y el receptor sino que también son leídos por los hosts intermedios. Un firewall puede ser configurado para permitir o rechazar el paso desde o hacia un puerto determinado.

Cuando en un host existe una aplicación que está esperando tráfico en un puerto dado (“escuchando”), se dice que el puerto esta “abierto”, si no hay aplicación que escuche en un puerto, éste está “cerrado”. No se pueden establecer conexiones sobre puertos cerrados.

Tanto TCP como UDP utilizan puertos en sus paquetes, sin embargo, ambos protocolos los utilizan de forma diferente.

UDP:

Todo el multiplexado y el demultiplexado entre el software UDP y los programas de aplicación ocurre a través del mecanismo de puertos. En la práctica, cada programa de la capa de aplicación debe negociar con el sistema operativo para obtener un puerto antes de poder transmitir un datagrama UDP.[COMER96]

Una vez negociado el puerto, este “pertenece” a la aplicación y todo datagrama que llegue destinado a ese puerto será entregado a esa aplicación.

TCP:

Los puertos en TCP son mucho más complejos que en UDP. Un numero de puerto no corresponde a un objeto, de hecho TCP fue diseñado sobre la

“Abstracción de Conexión”, en la que los objetos que se van a identificar son conexiones de circuito virtual y no puertos individuales.

El TCP utiliza la conexión, y no el puerto de protocolo, como abstracción fundamental; Las conexiones se identifican por medio de un par de puntos extremos. [COMER96]

En TCP un “punto extremo” es un par de números enteros (Host, puerto), en donde host es el IP de un host y puerto es un número de puerto TCP en dicho host. La conexión en TCP se define como un par de estos “puntos extremos”. Lo que permite que los mismos puertos sean usados por distintas conexiones.

Firewall:

“Existió un tiempo en que una computadora podía navegar en Internet por años sin ser atacada jamás, hoy en día, se necesitan sólo unos pocos minutos de conexión para que una máquina se encuentre bajo ataque (y esto es considerando únicamente aquellos ataques sin destinatarios específicos)”. [FAR-VEN06]

El crecimiento desmesurado de Internet ha resaltado las deficiencias en materia de seguridad de los sistemas de interconexión existentes. Rápidamente se determinó que el uso de un sistema para filtrar la información que llega a la interfaz de red disminuía drásticamente la información que se difundía sobre el sistema. Estos sistemas de filtrado se llaman firewalls.

Steven M. Bellovin y William R. Cheswick [BELL94] definieron 3 reglas a tomar en cuenta cuando se diseña un firewall:

1. Todo el tráfico, entrante y saliente, debe pasar por el firewall.

2. Sólo el tráfico que está definido en un conjunto de reglas locales puede pasar a través del firewall.

3. El firewall no debe ser penetrable (no debe contener vulnerabilidades que permitan penetrarlo).

Se pueden definir dos filtros en los firewalls, los que filtran el tráfico entrante (ingress filter) y los que filtran el tráfico saliente (egress filter).

Existen muchas variedades de firewall, cada variedad difiere de otra en cuanto al tipo de filtro y a la información de la cual dispone; En resumen existen los filtros de paquetes (Packet filter), los filtros de paquete con estado (Stateful packet filter), los filtros por aplicación (Aplication filters), puertas de enlace en circuito (Circuit Gateways), y firewalls distribuidos (distributed firewalls). Para este trabajo, nos interesa ver con mas detalle los filtros de paquete con estado.

Los filtros de paquete por lo general no tienen información de los protocolos de capa de aplicación, sino que, simplemente, la información que tienen sobre el paquete proviene de las cabeceras. Estos filtros toman decisiones basadas en las cabeceras de capa de enlace, red y transporte que tienen disponibles. Cuando los filtros de paquetes además manejan estado, les permite asociar paquetes con conexiones, lo que permite establecer reglas basándose en conexiones autorizadas.

La manera en la cual son asociados los paquetes con la conexión depende en gran parte de la implementación y sobre todo del protocolo de transporte que se usa. Por ejemplo en TCP, una implementación parcial de la máquina de estados TCP basta para saber cuando se crean y se cierran conexiones. Filtros con estado más sofisticados pueden hacer uso de rastreo de conexiones por medio de la información en las cabeceras de TCP como la Ventana y el número de secuencia. Para protocolos no orientados a conexión, como UDP, estos filtros pueden asociar los paquetes con una dupla de “fuente/origen” (probablemente con puerto), y asignarle un tiempo de vida a la conexión, la cual caducaría cuando se cumpla dicho tiempo si no ha habido actividad entre los puntos de la conexión.

La mayoría de los sistemas operativos modernos contienen algún tipo de “stateful packet filter”, como iptables en Linux.

Criptografía:

La Criptografía (del griego krypt´os: escondido y gr´aphein: escribir), es el arte de la transformación de datos a una forma no-legible, para ser descifrados sólo por aquellos que conozcan la clave secreta.

La criptografía nos permite proteger nuestra información de varias maneras para cuando la queremos transmitir sobre un medio o almacenar digitalmente. La confidencialidad es de suma importancia cuando se trata de transmitir o almacenar información “sensible” sobre un medio. [JEAN06]

Hoy más que nunca, con el auge de las telecomunicaciones sobre redes públicas, cuando se trata de información sensible, la criptografía nos provee la confidencialidad que necesitamos.

Servicios de criptografía:

La criptografía moderna se utiliza por lo general para proveer los siguientes servicios [DeGraaf07]:

- Autenticación: Seguridad de que la identidad de una entidad es la que dice ser. (Por ejemplo: username, password).
- Control de acceso: Prevención de acceso no autorizado a recursos.
- Confidencialidad de datos: Protección de los datos a ser vistos por entes no autorizados.
- Integridad de Datos: Seguridad de que los datos recibidos son los mismos que fueron originalmente transmitidos.
- No-repudiación: Impedir que el autor pueda negar su obra.

Todos estos servicios se ofrecen mediante el uso de los algoritmos de encriptado. Estos algoritmos pueden ser divididos en dos tipos:

- Algoritmos de criptografía simétrica.
- Algoritmos de criptografía asimétrica.

Los algoritmos también pueden ser clasificados según como manejan la información a procesar. Si estos utilizan la información a procesar como un flujo continuo, son llamados cifradores de flujo (stream cyphers), si en cambio dividen la información en bloques de tamaño predeterminado, y luego proceden a procesar cada bloque, entonces son conocidos como cifradores en bloque (block cyphers).

La criptografía simétrica usa una clave compartida para encriptar y desencriptar la información. Se tiene la misma clave para encriptar o desencriptar (la misma o una clave fácilmente deducible partiendo de la otra), que es un secreto compartido entre los entes autorizados. [JEAN06]

La criptografía simétrica se usa por lo general para mantener la confidencialidad de los datos, intenta asegurar que el texto cifrado no podrá ser descifrado sin la clave usada para encriptarlo.

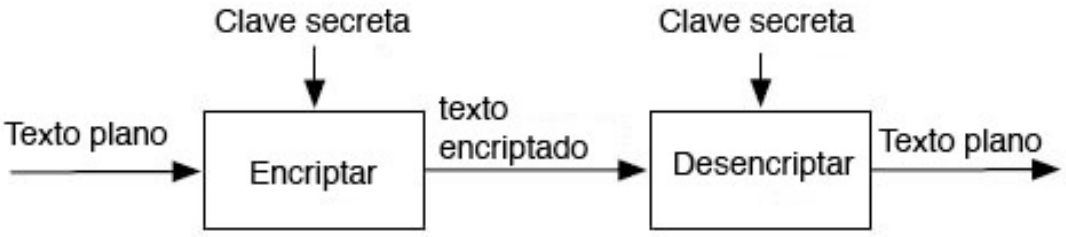

Fig. 3.2.1 Criptografía simétrica.

Esta criptografía es muy rápida y se puede implementar fácilmente por hardware (se embebe o integra la clave en el hardware). [ERIC03][JEAN06]

Los cifradores por bloques procesan los datos en “bloques” de texto de tamaño fijo (por ejemplo 256 bits). Los cifradores por bloques se usan en una variedad de “modos” como el Electronic Codebook (ECB), Counter (CTR) y Cypher Block Chaining (CBC), estos modos difieren entre sí primordialmente en la

manera en la cual manejan el cifrado de mensajes de tamaño mayor que el tamaño del bloque. [DeGraaf07][JEAN06]

CBC por ejemplo, uno de los modos más usados, aumenta la seguridad de un cifrador por bloques haciendo que cada bloque dependa de los mensajes anteriores. Cada nuevo texto cifrado depende del texto en plano actual y el bloque cifrado anterior. Este mecanismo permite entonces que dos bloques idénticos en un mensaje no se cifren de la misma manera, dificultando así el cripto-análisis del algoritmo de cifrado. [ERIC03][JEAN06]

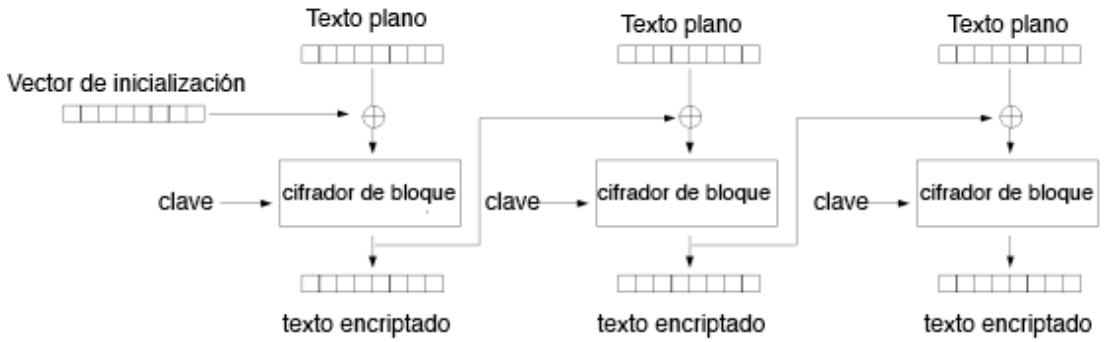

Fig. 3.2.1 Criptografía en bloque (ejemplo modo CBC) [ERIC03](reproducción)

Seguridad en Capas:

Cuando un sistema es vulnerable, se tiene que tomar el tiempo de analizar cuales son sus puntos débiles para así minimizarlos. En el caso de este trabajo de grado, la debilidad en el sistema se debe al diseño original de los protocolos de internet, en donde no fue tomada en cuenta la seguridad.

Los protocolos de conexión muchas veces permiten que se intercambie información aun antes de establecer formalmente la conexión (autorizada y/o autenticada). Esto permite muchas veces determinar información sobre el sistema remoto, lo cual aumenta el riesgo para el sistema.

Uno de los enfoques usados para aumentar la seguridad en un sistema, es el de seguridad en capas. Este enfoque consiste en agregar una capa de protección al sistema, manteniendo las capas subyacentes intactas.

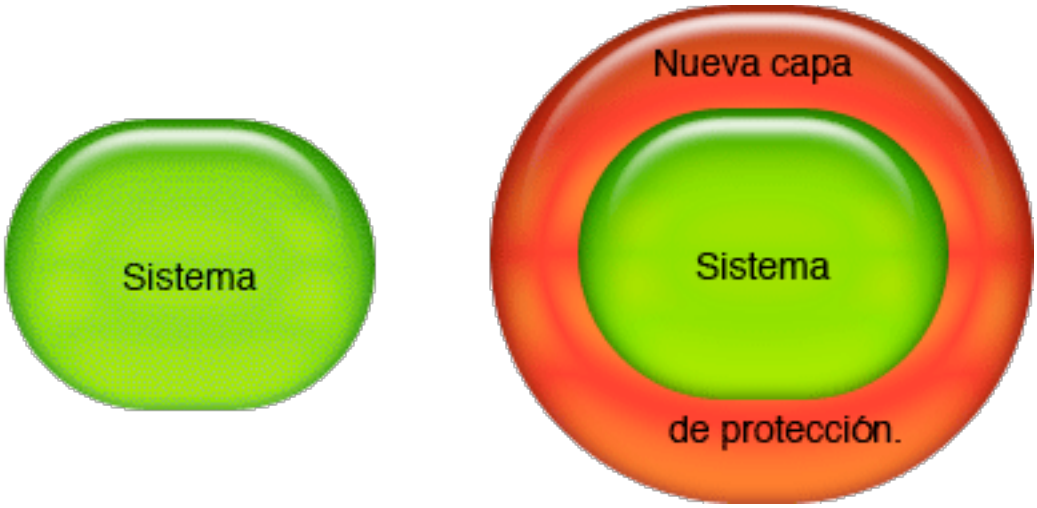

Fig. ## Seguridad en capas.

Esto permite aumentar la seguridad de un sistema sin tener que realizar cambios importantes en el sistema y al menor costo.

Autenticación sigilosa:

Existen dos tipos de sistema de autenticación sigilosa, Port Knocking y su sucesor Single Packet Authentication.

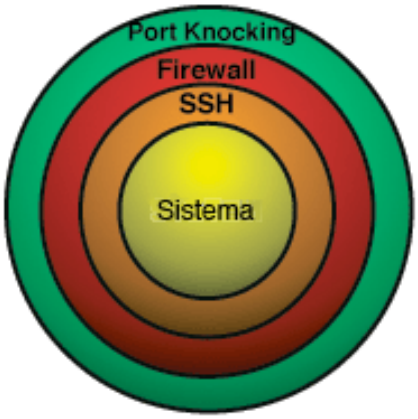

Fig. ## Port Knocking como capa de seguridad en capas.[JEAN06]

Port Knocking y SPA no intentan reemplazar ningún tipo de sistema de seguridad existente, sino que intentan agregarle una capa de seguridad extra a los mismos. Esto se basa en el proceso conocido como “seguridad en capas” en la cual se establecen niveles o etapas en la seguridad, y el vencimiento de un nivel no implica control total del sistema. [DeGraaf07][JEAN06] (Ver Fig. ##).

Básicamente ambos esquemas agregan una capa de seguridad nueva (sin quitar o modificar la anterior) que está compuesta de dos partes: Ocultamiento y Autenticación.

Los servicios que puede brindar un host son vulnerables a ataques de muchos tipos, y la mayoría de las veces, las vulnerabilidades encontradas y atacadas, son producto de errores en el código. Lo que PK y SPA intentan hacer con el ocultamiento, es esconder la existencia de un servicio en el host, exceptuando a los usuarios que se autentiquen en esa capa.

Este proceso de autenticación también pudiera revelar información sobre el servidor y allí está la clave de los sistemas que realizan autenticación sigilosa, en

estos, el proceso de autenticación se puede disimular con el tráfico común de las redes y es una autenticación en un solo sentido: El servidor no responde. C a b e destacar que autenticarse en esa capa solamente da acceso a la capa inferior, en la cual posiblemente se deberá volver a autenticar. Ambos sistemas están compuestos de tres componentes :

- Firewall: El firewall es el componente clave. Este firewall debe estar configurado para hacerle DROP a todos los paquetes que recibe. Es importante que sea DROP y no REJECT ya que con REJECT se puede enviar un mensaje de error de vuelta al host, en cambio que con DROP simplemente no hay respuesta. Esto permite que el host no pueda ser detectado por las técnicas comunes de detección (ping, barridos de puerto). La gran diferencia entre el DROP y el REJECT es que uno avisa de la existencia de un host en esa dirección, mientras que el otro no da avisos de la existencia, y hace imposible determinar que algún host con algún servicio se encuentra allí.

- Servidor pasivo: El servidor pasivo debe monitorear todo el tráfico que llega a la interfaz que está siendo protegida por el firewall. Esto lo puede hacer mediante análisis de logs o escuchando los paquetes directamente en el cable, con la ayuda de librerías como libpcap.

- Cliente: Utilidad para autenticarse en el servidor PK para poder tener acceso a los servicios que este protege.

La autenticación en estos sistemas se hace mediante mensajes codificados con un formato predeterminado, en los cuales hay información de autenticación, y por lo general algún comando que le indica al servidor lo que debe hacer al recibir una autenticación válida. Por lo general, la acción a realizar es abrir un puerto en el firewall para permitir una conexión entrante.

Port knocking y SPA no son inmunes a fallos; sus fallos pueden ser clasificados en dos tipos :

- Fallo cerrado : En el caso en que no se concrete la secuencia, o por alguna razón el servicio falle, el servicio subyacente sigue estando oculto, no se puede tener acceso al mismo pero sigue siendo seguro. [DeGraaf07] Esto básicamente es una denegación de servicio causada por un servicio interno.

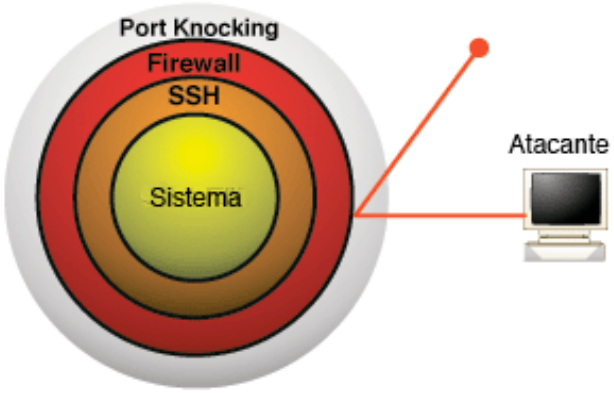

Fig. ## Fallo cerrado en autenticación sigilosa. [JEAN06]

- Fallo abierto: El sistema falla y deja abierto uno o más puertos. La ventaja de la seguridad en profundidad nos permite tolerar este tipo de fallos, ya que en este escenario, el sistema es tan vulnerable como si no estuviera el sistema de autenticación sigilosa.

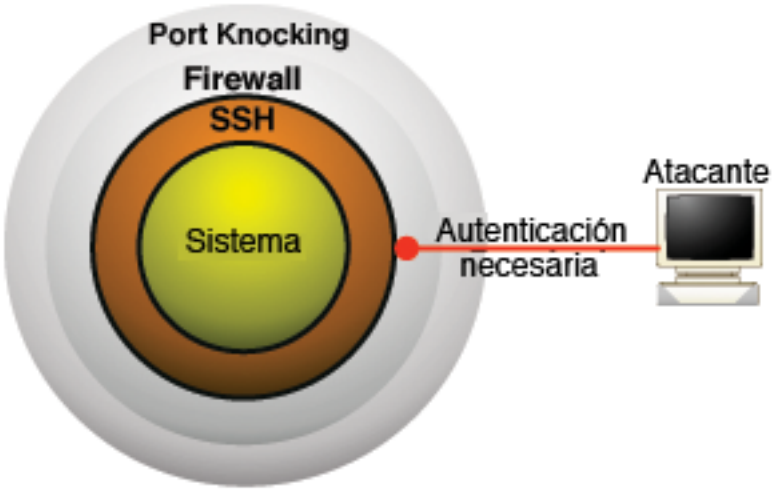

Fig.## Fallo abierto en autenticación sigilosa.

Diseño de la propuesta:

Descripción del problema:

Tomar el control de equipos en internet se ha vuelto una empresa muy lucrativa para los usuarios malintencionados [KAP08] y por esto, los ataques a los equipos conectados en internet son casi continuos [VAR-VEN06]. Configurar adecuadamente un sistema para resistir estos ataques no es simple, y mucho menos para el usuario promedio de la tecnología, lo que hace que millones de equipos conectados sean atacados con éxito, agravando la situación inicial.

La mayoría de estos ataques pueden ser fácilmente prevenidos con las configuraciones adecuadas, como por ejemplo la instalación de un sistema de autenticación sigilosa como PK y SPA, cuyos beneficios son evidentes para reducir la efectividad de los ataques continuos en los sistemas.

El problema con estos sistemas de autenticación es que, por lo general, son difíciles de configurar y no proveen mucha flexibilidad en cuanto a las acciones que pueden realizar, lo que los hace menos atractivos para el público general.

Ademas, muchas veces se necesita mantener el nivel de conectividad libre que una red no protegida permite, y por eso, simplemente establecer reglas de firewall estrictas no es suficiente.

El problema entonces es aumentar la protección de los servicios expuestos a Internet, sin configuraciones difíciles, manteniendo la posibilidad de conectividad tal como se tiene en una red poco protegida.

El sistema actual (ver Fig. ##) es un sistema totalmente desprotegido, en el cual todos los factores interactúan directamente con el sistema permitiendo así mas posibilidades de ataques.

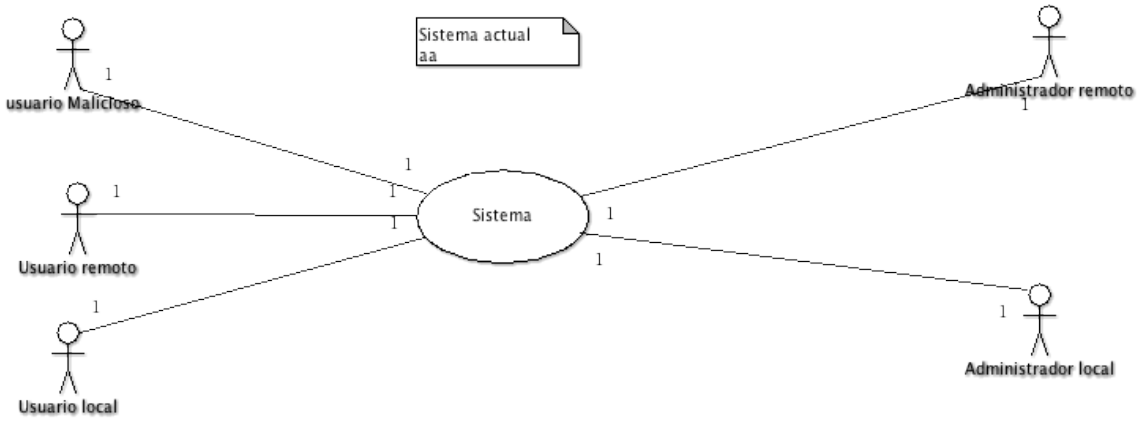

Fig. ## Sistema Actual.

Una solución para este problema de exposición directa es integrar en el esquema un firewall con reglas estrictas que restringen completamente el tráfico entrante (ver Fig. ##). Esto permite que solamente los actores internos, que están autorizados por las reglas del firewall, interactúen con el sistema y los servicios. Esto implica que estas reglas estrictas impiden que entes externos tengan acceso al sistema. Sin embargo, esto también excluye a ciertos actores que deberían poder interactuar con los servicios que ofrece el sistema, incluyendo los administradores remotos y los usuarios remotos.

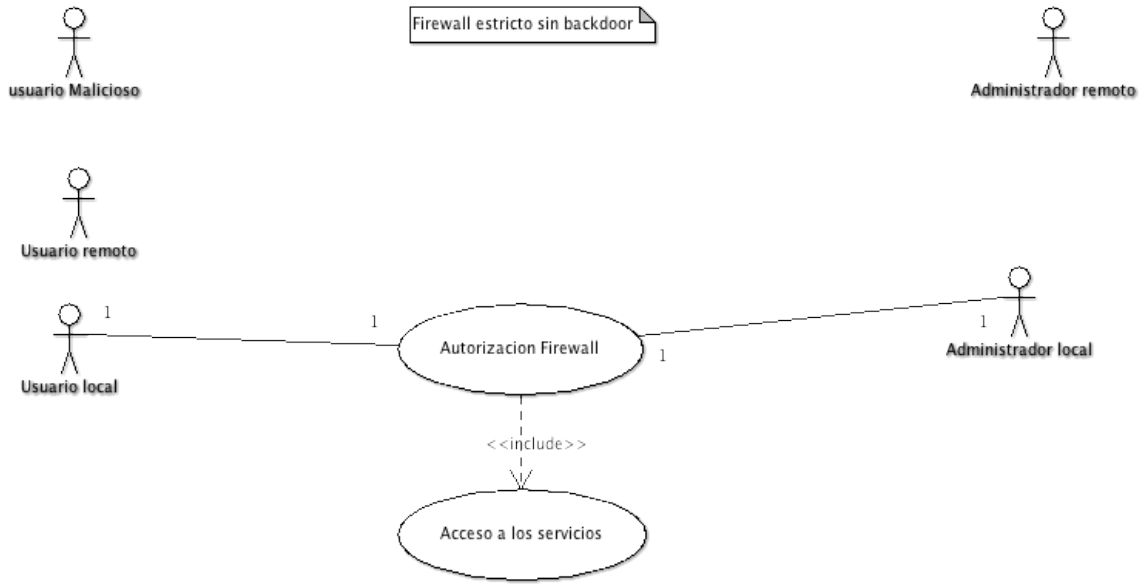

Fig. ## FIrewall con reglas estrictas.

Esta situación inhibe entonces las capacidades del sistema, lo que, para ciertas organizaciones y en ciertos casos, no es una opción viable.

Se dispone entonces de otro esquema, muy similar al anterior sólo que con una modificación que básicamente deja una puerta trasera abierta para permitir conexiones remotas (Ver Fig. ##). La puerta trasera puede ser administrativa, en el sentido en el que solamente un administrador del sistema la usa para realizar tareas administrativas en el servidor, o puede ser general, como un servicio público para que los usuarios remotos puedan tener acceso al sistema. El problema con este esquema, que naturalmente es más seguro que el esquema abierto, es que la puerta trasera podría también ser utilizada por un usuario malicioso para lanzar ataques.

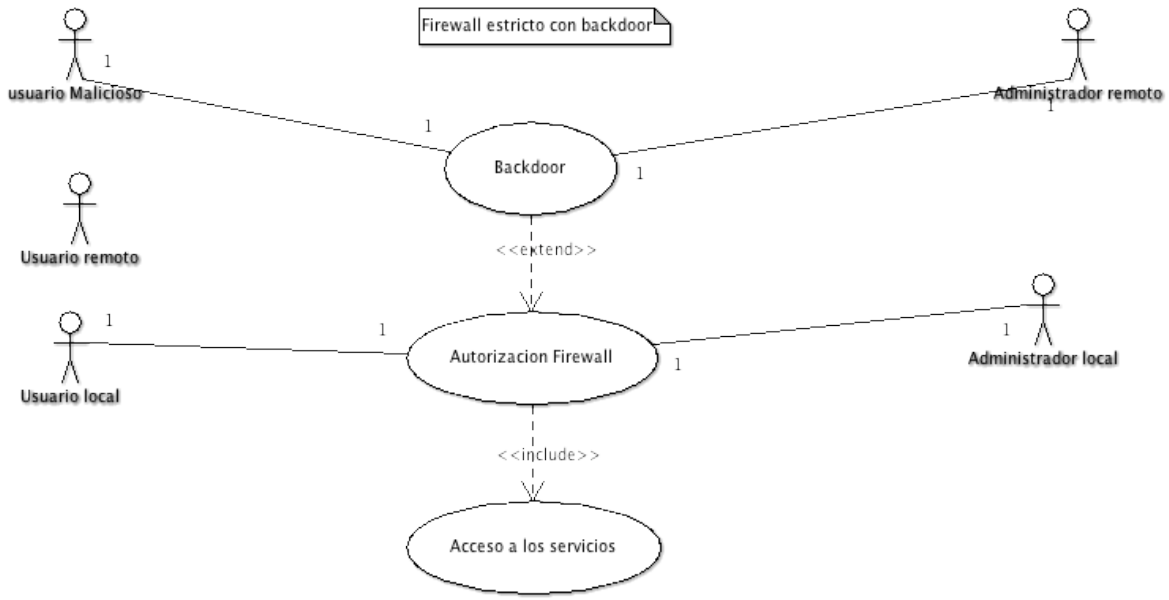

Fig. ## FIrewall estricto con Backdoor administrativo.

Descripción de la idea:

Una solución:

Los esquemas anteriores todos comparten el mismo problema: son estáticos, es decir que no pueden ser modificados. Los esquemas estáticos están fundamentalmente errados desde la fase de diseño, ya que es imposible contemplar todos los escenarios a los cuales se va a tener que enfrentar el sistema.

Los firewall permiten establecer reglas de acceso muy específicas, como por ejemplo autorizar el acceso al usuario autorizado conectándose desde un punto específico y al mismo tiempo denegar el acceso a un usuario no autorizado que se intenta conectar desde otro punto.

Esto implica que se puede tener un esquema en donde las reglas son estrictas para unos y abiertas para otros. El problema es que la naturaleza del usuario remoto implica que este tiene una alta probabilidad de trasladarse de localidad, lo que hace que las reglas estáticas que le permiten conectarse sean inútiles.

La solución a estos problemas está en implementar un sistema de autenticación sigilosa, que permite a un usuario autorizado (validado por medio de la autenticación) conectarse a un sistema que goza de la seguridad que brinda un firewall con reglas estrictas sin la vulnerabilidad de poseer una puerta trasera. Esto es posible porque los sistemas de autenticación sigilosa modifican dinámicamente las reglas del firewall, para así brindarle acceso al usuario autenticado a los servicios del sistema (ver Fig. ##).

Componentes

Un sistema de autenticación sigilosa tiene varios componentes, los cuales enumeraremos a continuación.

El primer componente es un firewall limítrofe, al cual se le delega toda la seguridad del sistema. Este firewall (packet filter) debe estar configurado para

bloquear todos los paquetes y datagramas que lleguen desde Internet. Esto proveerá la protección necesaria y deseada. Además de bloquear los paquetes, debe ser configurado para que no envíe respuesta alguna al emisor del paquete, de esta manera se proveerá el ocultamiento del servidor: sin respuesta de ningún tipo, un usuario malicioso no puede descubrir la presencia del servidor.

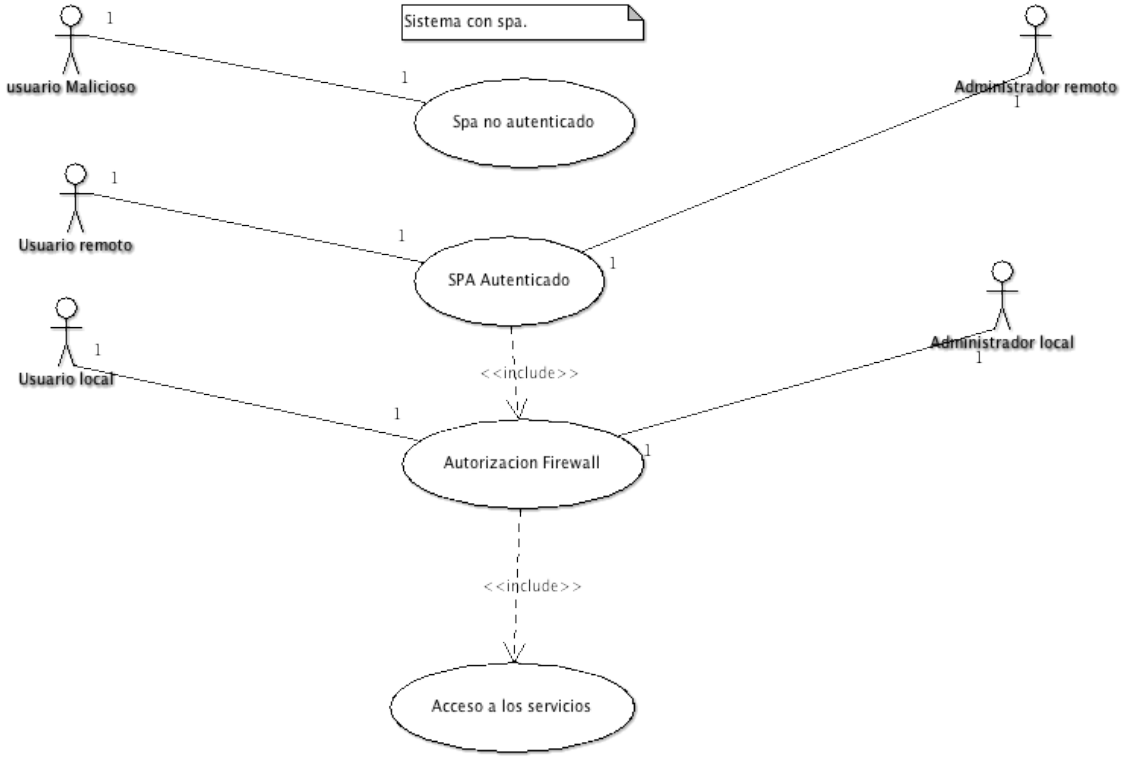

Fig. ## Sistema con SPA.

El segundo componente es el servidor de autenticación. Este es un servicio que debe estar ejecutándose en el equipo que sirve de firewall. Este servicio debe hacer la captura de bajo nivel de los paquetes que llegan a la interfaz de red, antes de que el firewall los descarte. Esto debe ser hecho a bajo nivel porque no hay librería de alto nivel que pueda capturar el paquete antes de que este sea procesado por el firewall. Ademas de capturar los paquetes, este servicio debe verificar el contenido de la carga útil de los mismos para buscar un mensaje de autenticación válido. Cuando un mensaje de autenticación válido es recibido, el servidor de autenticación debe modificar las reglas del firewall para autorizar las conexiones entrantes para el cliente que lo envió. Otra de las funciones que debe

proveer el servidor de autenticación es el rastreo de conexiones activas y conexiones terminadas o vencidas. Esto es para evitar dejar el firewall abierto, una vez que el usuario autenticado ha terminado su sesión, así que es parte del trabajo del servidor de autenticación proveer un mecanismo de “roll back” de las acciones tomadas sobre el firewall para poder volver al estado inicial.

El tercer y último componente del sistema es el cliente de autenticación. Esta es una aplicación que permite al usuario autenticado enviar el paquete bien formado que lo autenticará en el servidor.

Con este esquema se obtienen los beneficios de un firewall con reglas estrictas, pero se mantiene la libertad de un sistema abierto, con sólo añadir una capa extra de protección y autenticación (ver Fig. ##).

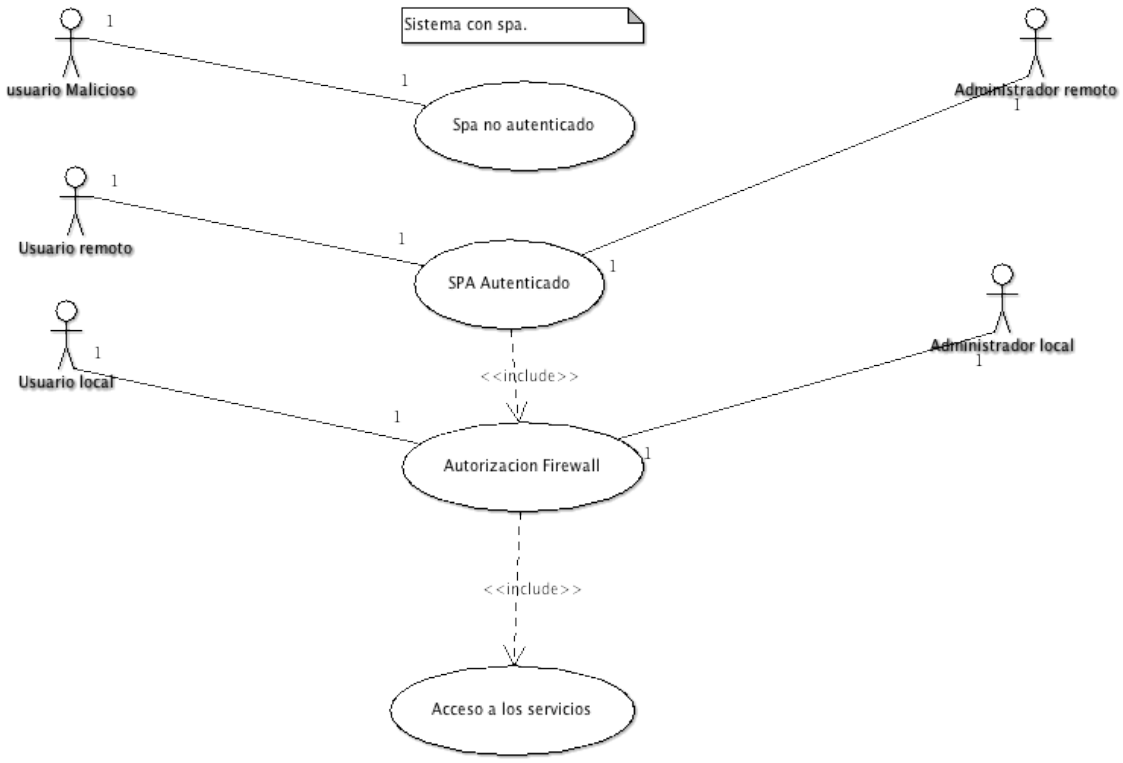

Fig. ## Sistema con SPA.

Este diseño implica que se debe desarrollar un cliente de autenticación y los componentes del servidor de autenticación.

Justificación:

La idea de este trabajo es demostrar que se puede proveer una capa extra de seguridad a un sistema existente, sin perder la posibilidad de conectividad desde el exterior, y sin tener que modificar los servicios y aplicaciones en uso. Los sistemas de autenticación sigilosa son sistemas que ya han sido probados y sus beneficios han sido demostrados [Khak-Chao07], la autenticación sobre puertos cerrados provee una capa de seguridad que es análoga a la de un firewall con reglas muy estrictas, y la posibilidad de establecer reglas de firewall dinámicas brinda la flexibilidad de un sistema abierto.

En donde fallan los sistemas existentes es en la facilidad y flexibilidad de las acciones post-autenticación y en la configuración de las mismas.

En este trabajo se busca implementar un sistema de autenticación sigiloso completo, sin amarrar al usuario a una tecnología particular.

El cliente se desarrollará en JAVA, ya que es un lenguaje multi-plataforma con el cual se puede implementar una aplicación que cumpla con el protocolo diseñado para esta aplicación (el cual será explicado mas adelante). Esto permite ofrecer el servicio a clientes que utilicen distintas plataformas, manteniendo así la flexibilidad que queremos caracterice la aplicación.

El servidor de autenticación estará alojado en un equipo que tenga instalado el sistema operativo linux, ya que es un software libre (de código abierto) y gratuito, muy bueno para alojar servicios web. Pero esto no es un requisito, el servidor podrá ser compilado en cualquier sistema que tenga un compilador GCC (de GNU), y la librería de captura de paquetes PCAP.

Además de ser un sistema operativo muy robusto, todos los sabores de linux hoy en día tienen integrado NETFILTER en el kernel y su aplicativo IPTABLES, que juntos conforman un excelente filtro de paquetes con estado, así que usaremos este firewall en nuestra implementación. Sin embargo, para mantener nuevamente la idea de flexibilidad, se pudiera usar otro firewall si se quisiera, con pocas modificaciones al código.

Para la implementación del servidor de autenticación se escogió el lenguaje de programación C. Este lenguaje es muy eficiente y debido a que se van a estar escuchando y parcialmente procesando todos los paquetes que transitan por la interfaz de red, se necesita un lenguaje que tenga un muy alto rendimiento para que el rendimiento del sistema no se vea afectado por el nuevo servicio [DEB-BENCH].

La limpieza de las conexiones vencidas se podía hacer de dos maneras distintas, con un hilo en el servicio de autenticación o con un programa externo.

Para maximizar la flexibilidad, la limpieza se implementó como parte de un programa externo que interactúa con las bitácoras de conexiones, realizadas por el servicio de autenticación, para poder cerrar las conexiones vencidas. Esto permite que, si no se desea usar el script provisto, se pueda implementar otro que lo remplace. Para este trabajo, se implementó un script en PHP-CLI que verifica si las conexiones han sobrepasado cierto tiempo (configurable), si es así entonces las revierte en el firewall. Este script es agregado en un CRON para que se inicie cada cierta cantidad de minutos (tiempo que debe ser igual o menor al tiempo de vencimiento de las conexiones).

Como librería de captura de paquetes se utilizó libpcap, que es una librería de código abierto que ha sido utilizada en aplicaciones conocidas como nmap, ethereal (wireshark).

La comunicación entre el cliente y el servicio de autenticación incluye información sensible, que si es interceptada por un usuario malicioso pudiera comprometer toda la seguridad del sistema, así que toda comunicación de autenticación debe viajar encriptada. Como la idea es minimizar el tráfico que entra y sale del servidor, se utiliza un algoritmo de criptografía simétrica (para evitar tener que hacer intercambio de claves). El algoritmo escogido es Blowfish, que es un algoritmo de criptografía simétrica con bloques de 8 bytes y una clave de 128 bits. Para el cliente se utilizó la librería criptográfica bouncycastle [BC], y para el servicio de autenticación se usaron las librerías criptográficas de openssl. Ambas librerías hacen una implementación estándar del algoritmo blowfish lo que

las hace compatibles. (ver anexo 2 para instrucciones de instalación de todas estas librerías).

Protocolo:

El cliente y el servicio de autenticación tienen que comunicarse entre ellos. Esta comunicación debe poder hacer dos cosas en concreto; Autenticación y comando; además debe hacerlo en la menor cantidad de tiempo posible y de manera unilateral. Esto es porque el servidor no debe enviarle respuesta positiva o negativa al cliente, para mantenerse lo más oculto posible.

Así que en el mensaje de autenticación deben enviarse los datos necesarios para realizar la autenticación. El esquema de autenticación que se usa en este sistema es nombre de usuario y la contraseña. Estos nombres de usuario y contraseña están almacenados en un archivo llamado authusers.spa en la carpeta configs del servidor de autenticación.

El formato de este archivo es simple :

Una linea por usuario.

Nombre de usuario = contraseña.

Cuando un cliente envía su datagrama debe siempre incluir esta información de autenticación. El datagrama tiene el siguiente formato :

username::password::<marca de tiempo>::<instrucción>

Más sobre el campo <instrucción> en los detalles de la implementación.

Detalles de la implementación:

En este trabajo se implementó un sistema de autenticación sigilosa, basándose en los sistemas de *single packet authentication*, por ende, se implementaron varios componentes del sistema para que el mismo funcionara correctamente.

Los sistemas de autenticación sigilosa son sistemas cliente/servidor (ver Fig. ##), como tal, se tienen que implementar tanto el componente de cliente como el de servidor. La gran diferencia radica en que el servidor no provee respuesta otra que la acción desencadenada por una autenticación válida.En este capítulo se detallarán las partes de la implementación más relevantes.

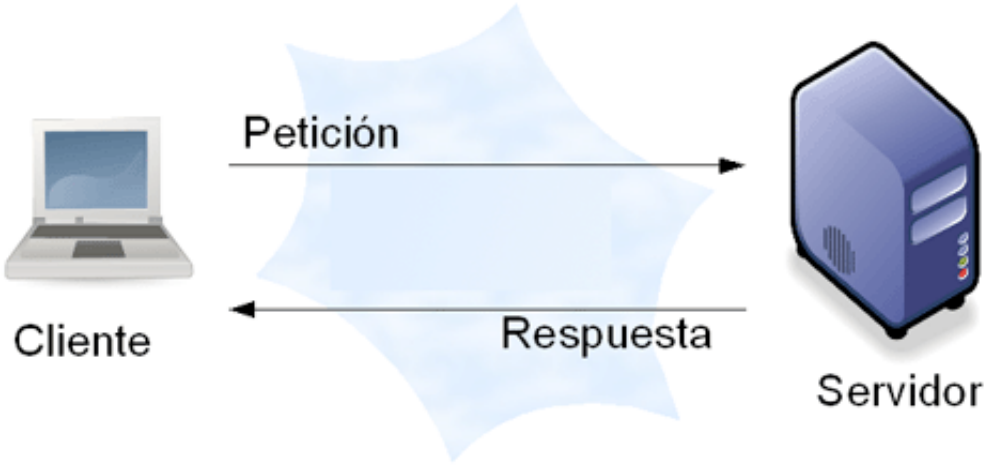

Fig. ## Esquema Cliente/Servidor.

Uno de los problemas con los sistemas de autenticación sigilosa existentes, es que la mayoría de ellos no permite gran variedad de acciones post-autenticación. Por lo general, simplemente se contentan con abrir un puerto deseado y esperar que se venza el tiempo para cerrarlo. Los que si permiten un poco mas de rango de acciones, lo hacen con muchas restricciones y por lo general por medio de configuraciones muy complicadas. La idea de este trabajo es hacer una implementación que sea lo más flexible posible y que permita mayor rango de funcionalidad.

ActionTypes, distintos tipos de acciones

Con el fin de realizar una implementación que sea lo más flexible posible, se establecieron claramente los tipos de acciones que va a poder manejar el sistema al recibir un mensaje autenticado. Para el propósito de este trabajo se definieron los que se consideraron necesarios. Cada uno de estos tipos de acción se llama ActionType y difieren entre ellos en cuanto a la cantidad y al tipo de parámetros que aceptan. Cabe destacar que en los archivos de configuración, el lugar de los parámetros se establece usando la notación printf de C para las cadenas %s . Estos tipos de acciones se describen a continuación:

ActionType tipo 1:

El actionType tipo 1 es uno de los más simples en el sistema, es una acción sin parámetros. Luego de haber validado el nombre de usuario y contraseña que llegan en el AP, si se encuentra con un actionType de tipo 1, entonces el sistema sabe que debe ejecutar la acción que está en el archivo de configuración para la acción llamada.

Así, si el archivo de configuración tiene en su campo acción lo siguiente :

- `iptables -F`

Entonces se ejecutará exactamente como aparece. Esto da la flexibilidad de configurar acciones repetitivas en el servidor con un comando, permitiendo así realizar tareas de administración remota con el envío de un solo paquete por la red. No existe ni siquiera la necesidad de abrir el puerto y establecer la conexión, solamente con enviar un paquete autenticado que llame a una acción tipo 1 se pueden realizar muchas cosas rutinarias (como por ejemplo, limpieza de logs, reinicio de servicios etc.)

ActionType tipo 2:

Este tipo de acción permite definir en el archivo de configuración, un comando a ejecutar que acepta un parámetro en sí mismo. Este parámetro siempre será el IP del cliente, así que se puede ejecutar cualquier comando que requiera un IP. El ejemplos mas obvio sería insertar una regla nueva en el firewall exclusivamente para este cliente. Desde el lado del cliente esto no tiene ningún tipo de configuración, simplemente se envía el paquete solicitando una acción de tipo 2, y la definición de la acción que está en el archivo de configuración va a ser recorrida para insertar el IP del cliente.

De esta manera se puede tener en el archivo de configuración una acción definida de la siguiente manera:

- `iptables -A INPUT --source %s -j DROP`

y cuando se reciba un paquete autorizado solicitando esta acción, será ejecutado :

- `iptables -A INPUT --source <IP DE EL CLIENTE> -j DROP`

Se pueden entonces definir cualquier acción que utilice el IP del cliente como parámetro.

ActionType tipo 3:

Para permitir definir reglas menos estrictas y que pueden ser configuradas desde el cliente, se define otro actionType, esta vez, el actionType permite dos parámetros, uno que va a viajar en el paquete de autorización (puede ser user input) y otro que es el IP del cliente. De esta manera se pueden crear reglas un poco mas genéricas en el servidor, con el actionType 3 y su acción se definen dos parámetros a ser reemplazados luego.

Por ejemplo se pudiera tener :

- `iptables -A spa --dport %s --source %s`

Al enviar el paquete, el cliente debe incluir un parámetro en el campo instrucción de su AP. Por ejemplo en esta acción, el parámetro debe ser el puerto a abrir. Si por ejemplo quisiera abrir el puerto 80 (para conectarse al servidor web) pudiera enviar el 80 como parámetro y en el servidor se ejecutaría:

- `iptables –A spa –-dport 80 –-source <IP DEL CLIENTE>`

Esto permite entonces generalizar más en el servidor brindando más configuración en el cliente.

ActionType tipo 4:

Este brinda un nivel más de configuración en el cliente, y más generalización aun en el servidor. Permite 3 parámetros, el último de los cuales es el IP del cliente y los dos anteriores deben ser especificados en el campo instrucción del AP. Por ejemplo:

- `iptables –A spa –-dport %s –p %s –-source <IP DEL CLIENTE>`

Esto permite al cliente decidir si abre un puerto para él mismo, especificando el protocolo de transporte usado. Se puede entonces ejecutar una sola acción definida :

- `iptables –A spa –-dport 80 –p tcp –-source <IP DEL CLIENTE>`
- `iptables –A spa –-dport 53 –p udp –-source <IP DEL CLIENTE>`

ActionType tipo 5:

El actionType tipo 5 es muy parecido al tipo 2, un solo parámetro es permitido, pero a diferencia del tipo 2, este parámetro no es el IP del cliente, sino que es cualquier cosa que se envíe en el campo instrucción del AP. Se pudiera tener por ejemplo una acción definida de la siguiente manera:

- `iptables –F %s`

Entonces se pudiera enviar por el AP a cuál cadena de iptables se le debe hacer flush. Este tipo de acción no debería ser accesible (por medio del cliente) a

cualquier usuario, se pueden realizar muchas tareas privilegiadas con este actionType, como por ejemplo:

- `/etc/init.d/%s`
- `%s`

En el primer ejemplo se crea una regla que permite realizar cualquier acción con los servicios del sistema (enviando “apache2 restart” se puede reiniciar apache), y en el segundo ejemplo se crea un terminal de administrador virtual. Cualquier comando bash puede ser enviado en el AP y será ejecutado (hasta por ejemplo “init 0”).

ActionTypes tipo 6 y 7:

Estos dos actionTypes son simplemente acciones definidas de la misma manera, con 2 y 3 parámetros respectivamente en donde todos los parámetros viajan en el campo instrucciones del AP (el IP del cliente no es un parámetro en estos tipos).

ActionType tipo 8:

El actionType tipo 8 es el tipo de acción que le facilita al administrador del sistema la configuración de redirecciones que se ajustan al modo de operación número 2 del sistema implementado. Permite establecer, directamente en el archivo de configuración, una redirección de puertos hacia otro host (por medio de nateo de origen y destino). Estas reglas de firewall están compiladas en el código y son el único pedazo de código que habría que modificar para cambiar el firewall que se desea usar con el sistema. Para este trabajo, el tipo 8 se llama basic config.

Cualquier actionType que no sea identificado, será considerado de tipo 0 y descartado.

Archivos de configuración:

Todas las configuraciones del sistema se almacenan en una carpeta llamada “configs” en el directorio de la aplicación.

En el sistema existen dos tipos de archivos de configuración:

El primero es simplemente el listado de las duplas <nombre de usuario> y <contraseña> de los usuarios autorizados para enviar AP. Este archivo se llama authusers.spa, un ejemplo del cual se da a continuación:

```
cromestant=12345
alebe=54321
chris=poisa
chachopo=asdas
```

El segundo es el que define una acción para el servidor. En ese archivo se establece qué debe hacer el servidor al recibir esa acción. El nombre del archivo es el <número de acción>.spa y su contenido se describe a continuación:

```
<actionType>
<rollback>
<comando>
```

En donde <actionType> es el número de acción tal como lo definimos en la sección anterior, <rollback> es un 0 o un 1, simplemente se pone en 1 cuando es una regla de firewall y se desea que el sistema monitoree cuando se ha vencido el tiempo de la acción sin algún mensaje del cliente, en ese caso se genera una entrada en las bitácoras de conexiones para poder revertir la regla en el firewall, si está en 0 entonces el sistema no monitorea el tiempo vencido. <comando> es una cadena que contiene el comando a ser ejecutado con los contenedores de parámetros según el tipo que sea. Un archivo de configuración de acción completo puede verse así :

```
3
1
iptables -A spa -p tcp --dport %s --source %s -j ACCEPT
```

Existe una variación en los archivos de configuración de las acciones, y es para cuando se define una acción del tipo basic config. Este archivo no tiene

exactamente la misma configuración ya que no requiere de comando pero si requiere de otros datos. La estructura de un archivo basic config se tiene a continuación:

<8>
<rollback>
<ip destino>
<puerto>

En donde <8> es el número de basic config, siempre es 8, <rollback> es el mismo que en las configuraciones de las demás acciones, <ip destino> es la dirección IP hacia donde se va a realizar la redirección y por ultimo <puerto> es el puerto entrante donde se va a hacer la redirección. Un ejemplo de archivo basic config se tiene a continuación:

```
8
1
10.0.10.1
80
```

Bucle de captura (pcap loop):

La librería de captura de paquetes de red es el componente más importante de este sistema. Sin ella no se pudiera escuchar el tráfico de red sobre puertos cerrados. En el código, la sección más importante en relación a la captura es el bucle de captura. La librería permite establecerlo mediante la llamada pcap_loop en donde recibe por parámetros el descriptor de captura (handler de la interfaz de red), la cantidad de paquetes a capturar (si es -1 entonces siempre sigue), el apuntador a la función que procesará el paquete una vez capturado y por último un apuntador a los parámetros que se le quieren pasar a la función manejadora de paquetes.

```
pcap_loop(descr, -1, got_packet, NULL);
```

En el ejemplo, la función que va a manejar el procesamiento de los paquetes capturados es got_packet(). Esta función recibe el paquete en su forma mas cruda, con todas las cabeceras agregadas por las distintas capas. Las cabeceras Ethernet e IP no nos interesan para este trabajo así que hay que quitárselas a todos los paquetes entrantes. Para este efecto se crearon las estructuras de datos necesarias en C para poder extraer todos los datos de cabeceras que nos interesan.

Cabecera Ethernet [IEEE 802.3]:

```
struct sniff_ethernet {
  u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address */
  u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
  u_short ether_type;                     /* IP? ARP? RARP? etc */
};
```

Cabecera IP [RFC 791]:

```
struct sniff_ip {
        u_char  ip_vhl;        /* version << 4 | header length >> 2 */
        u_char  ip_tos;                /* type of service */
        u_short ip_len;                /* total length */
        u_short ip_id;                 /* identification */
        u_short ip_off;                /* fragment offset field */
        #define IP_RF 0x8000           /* reserved fragment flag */
        #define IP_DF 0x4000           /* dont fragment flag */
        #define IP_MF 0x2000           /* more fragments flag */
        #define IP_OFFMASK 0x1fff      /* mask for fragmenting bits */
        u_char  ip_ttl;                /* time to live */
        u_char  ip_p;                  /* protocol */
        u_short ip_sum;                /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
};
```

Una vez extraídas las cabeceras ethernet e IP, se procede a verificar qué tipo de protocolo de capa de transporte se utiliza: TCP o UDP. Para esta implementación los mensajes de autenticación serán transmitidos solamente sobre UDP, ya que no se requiere respuesta, ni creación formal de la conexión (three way handshake). Esto se hace mediante la verificación del campo protocol en la cabecera anterior :

```
      switch(ip->ip_p) {
            case IPPROTO_TCP:
                  printf("   Protocol: TCP\n");
                  break;
            case IPPROTO_UDP:
                  printf("   Protocol: UDP\n");
                  //marco la bandera para saber que es UDP.
                  flag =1;
                  break;
            case IPPROTO_ICMP:
                  printf("   Protocol: ICMP\n");
                  return;
            case IPPROTO_IP:
                  printf("   Protocol: IP\n");
                  return;
            default:
                  printf("   Protocol: unknown\n");
                  return;
      }
```

Entonces si el protocolo no es UDP simplemente se descarta el paquete. No tiene sentido procesar más el paquete si no es el que estamos buscando. En cambio

que si es UDP entonces se procede a extraer la cabecera UDP para su análisis, con esto se extrae la carga útil del datagrama.

Cabecera UDP [RFC 768]:

```
struct sniff_udp {
      u_short uh_sport;       /*UDP header source port*/
      u_short uh_dport;       /*UDP header destination port*/
      u_short udp_len;        /*UDP message length*/
      u_short udp_sum;        /*UDP checksum*/
      #define UDP_OFF(th)     (((th)->udp_len & 0xf0) >> 4)
};
```

El campo udp_len nos ayuda a conseguir el final de la carga útil del datagrama. Con esto se puede proceder a procesar el datagrama, esto incluye validar la autenticación y luego verificar cuál es la instrucción a ejecutar.

Criptografía:

Como la carga útil es encriptada en el cliente, al llegar al servidor debe ser desencriptada, para esto se llama al método do_decrypt desde el bucle de captura, de nuevo, si al desencriptar hay algún error, entonces el datagrama se descarta ya que no tiene el formato esperado.

El algoritmo criptográfico utilizado en este trabajo es Blowfish [Schnei], que es un algoritmo de criptografía simétrica para el cual aun no se ha encontrado cripto-análisis efectivo. Es un algoritmo que fue diseñado para ser rápido, lo que es muy atractivo para nuestra aplicación, y es gratuito y de implementación abierta. Blowfish puede usar claves de tamaño variable (de 32 bits a 448 bits) y es un cifrador por bloques. En nuestra implementación se utiliza blowfish con clave de 128 bits y tamaño de bloque de 64 bits en modo CBC [SEMINARIO].

En el cliente se utiliza una librería criptográfica llamada Bouncy Castle [BC] que implementa los algoritmos criptográficos más comunes de manera muy segura y con código abierto (Ver Anexo ## para detalles de implementación usando bouncy-castle).

Para el servidor se utilizan las librerías criptográficas de la suite openssl [http://www.openssl.org/], que son la base de las implementaciones de SSL en código abierto. Esta implementación es extensamente usada en el mundo y constantemente está siendo probada y avalada como una librería de alta calidad (Ver Anexo ## para detalles de implementación usando openssl).

Ambas librerías se apegan a los estándares de implementación de Blowfish y por ello son compatibles.

Log de rollback

Cuando una acción es cargada por el servidor de la aplicación, carga un parámetro llamado <rollback>. Si este campo está en 1, quiere decir que es una regla de firewall y que hay que vigilarla para poderla revertir si se vence el plazo establecido.

Esto se almacena en un archivo por cliente, dicho archivo se guarda en el servidor bajo un directorio en /var/spaUCV . Cada archivo guardado pertenece a un cliente, y se utiliza un solo timestamp para todas las acciones reversibles que haya activado este cliente.

El cliente debe, a cada cierto tiempo, enviar un AP especial que se llama “keep alive”, que simplemente le dice al servidor de actualizar el timestamp en ese archivo, de manera que así se vuelve a iniciar el tiempo de vida de las acciones para ese cliente.

En el servidor, existe un trabajo CRON, que se ejecuta a cada cinco minutos y verifica uno a uno estos archivos para revisar si alguno está vencido, de ser así, ejecuta los comandos inversos para revertir el firewall a su estado inicial.

```
if (($now-$ts)>$timeOut)
            {
                  while($action = fgets($fh)){
                  exec($action);
                  }
                  fclose($fh);
                  unlink($dir.$file);
            }
```

El código de este trabajo CRON fue realizado en PHP-cli por la simplicidad y longitud del código en ese lenguaje, pero puede en realidad ser implementado en cualquier lenguaje que se desee. Se puede ver el código fuente del script en el anexo número 1.

Pruebas y resultados

El propósito del sistema desarrollado en este trabajo especial de grado es proveer una nueva capa de seguridad al sistema existente. Esto implica que se debe solamente agregar un componente al sistema sin modificar los componentes anteriores, salvo las configuraciones del firewall.

Se buscaba proveer una capa de autenticación sigilosa al sistema, basada en un sistema de *Single Packet Authentication*, desarollado en el transcurso de este trabajo.

Se extendió más allá de esta funcionalidad de autenticación para proveer también las facilidades de administración remota de los servicios del servidor de autenticación sin que el administrador tenga que iniciar sesión.

Para demostrar que las metas del trabajo fueron logradas, se efectuaron las siguientes pruebas :

- Pruebas de funciones básicas.
- Pruebas de rendimiento.
- Pruebas de detección y barrido de puertos.
- Pruebas de resistencia a la denegación de servicio.

Todas estas pruebas, salvo las de funciones básicas, se realizaron tanto en el sistema con protección como en el sistema desprotegido para poder comparar los resultados , los cuales serán presentados a continuación.

Definición del ambiente de pruebas

El ambiente de pruebas utilizado intenta simular una situación común para una red privada que tiene conexión a Internet.

Se definen tres entes en el ambiente :

- Servidor público: es el servidor que tiene el firewall limítrofe del sistema, tiene dos interfaces de red, una conectada a la red privada y otra a la red pública, este sirve de “puente” entre la red privada y la red publica. Es en este host que esta funcionando el sistema SPA desarrollado en este trabajo de grado al cual llamaremos ‘servicio de autenticación’.
- Servidor privado : este es el host que pertenece a la red privada del sistema de pruebas, en el se encuentran servicios que son inaccesibles desde la red pública.
- Cliente : El cliente es un host que se encuentra conectado a la red pública, este cliente tiene que autenticarse para poder tener acceso a los servicios del servidor de autenticación o del servidor privado.

La dirección de red en la red privada es 10.0.10.0/24 , y la dirección de red para la red pública es 10.0.1.0/24.

El Cliente se conecta al servidor público por medio de un switch de red / Access point en la interfaz wlan1 del servidor público, y el servidor público se conecta al servidor privado por medio de su interfaz de red eth0.

El cliente en las pruebas es un equipo con el sistema operativo Macos X leopard 10.5.4, con java 1.5 instalado.

El servidor privado es un equipo con el sistema Macos X tiger 10.4.9 instalado, los servicios que corre son *openssh* y *apache2 webserver*.

El servidor público es un host que tiene instalado Ubuntu 8.04, versión estándar. Se utilizan las siguientes versiones :

- iptables v1.3.8.
- libssl v 0.9.8.
- libpcap v 0.8.
- gcc v4.2.3.
- GNU Make v3.81.

Pruebas de funciones básicas

Las pruebas de funciones básicas son simples pruebas que verifican que la funcionalidad propuesta por el sistema esté funcionando de verdad. Esto implica utilizar todas las partes del sistema para autenticar a un usuario remoto y luego ejecutar alguna acción en el servidor público para proveer alguno de los servicios.

Estas pruebas son de nueve funciones distintas, para las cuales se compiló el cliente. Aunque estas son todas las funciones que se configuraron para este trabajo, el sistema es capaz de configurar muchas más acciones como se describe en el capítulo “Detalles de la implementación” de este documento.

Flush de las tablas del firewall

La primera acción definida en la aplicación cliente es un comando administrativo para purgar algunas tablas del firewall y revertir el bloqueo por defecto del firewall. En concreto, se hace flush a las tablas INPUT y a la tabla definida para la aplicación, llamada spa y por último, se establece la politica por defecto de la tabla INPUT a ACCEPT.

En el servidor se ejecuta :

```
Case 1: Ejecutando
iptables -F INPUT
iptables -F spa
iptables -p INPUT ACCEPT
```

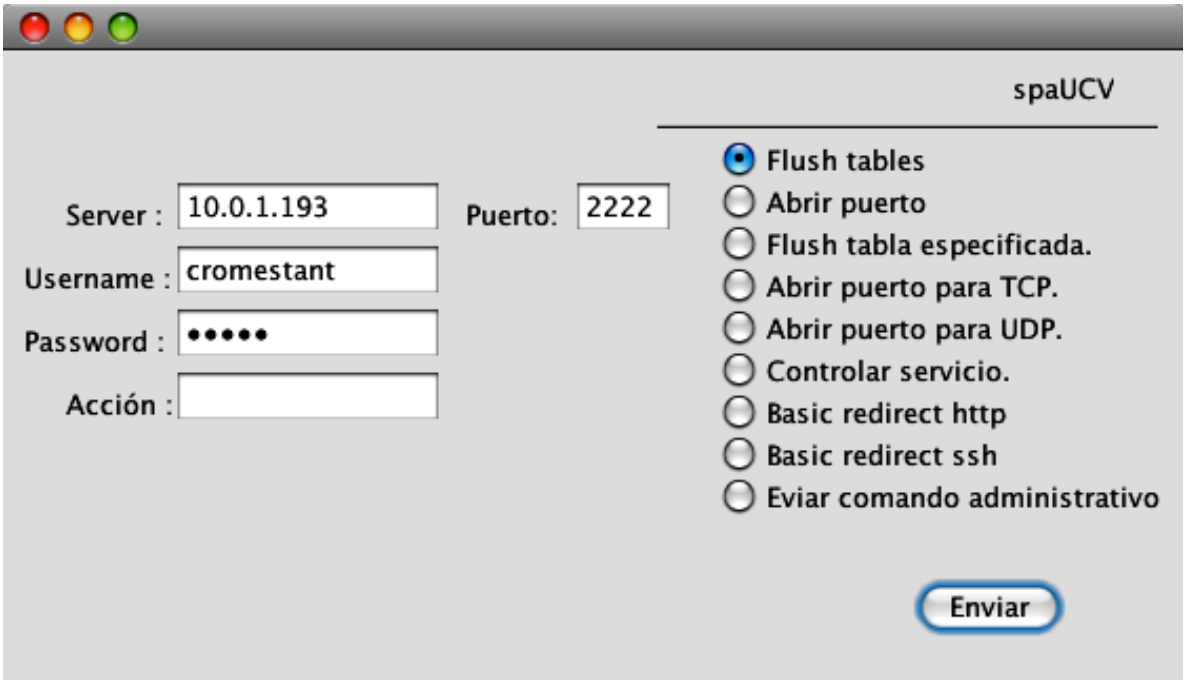

Fig. ## Cliente Acción 1.

Abrir puerto

Esta acción, como su nombre lo indica, abre un puerto en el firewall del servidor público. Esto permite acceder a los servicios de este servidor. En el campo Acción de la interfaz gráfica del cliente se específica el o los puertos que se desean abrir (los rangos de puertos son válidos, siempre y cuando se apeguen a las notaciones establecidas por iptables, ver `man iptables` para mas detalles).

Un ejemplo de esta acción , para abrir el puerto 80 , en el servidor se ejecuta :

```
Case 3: Ejecutando iptables -A spa -p tcp --dport 80 --source 10.0.1.199
-j ACCEPT
```

Abriendo el puerto 80 solamente para el cliente que lo solicitó.

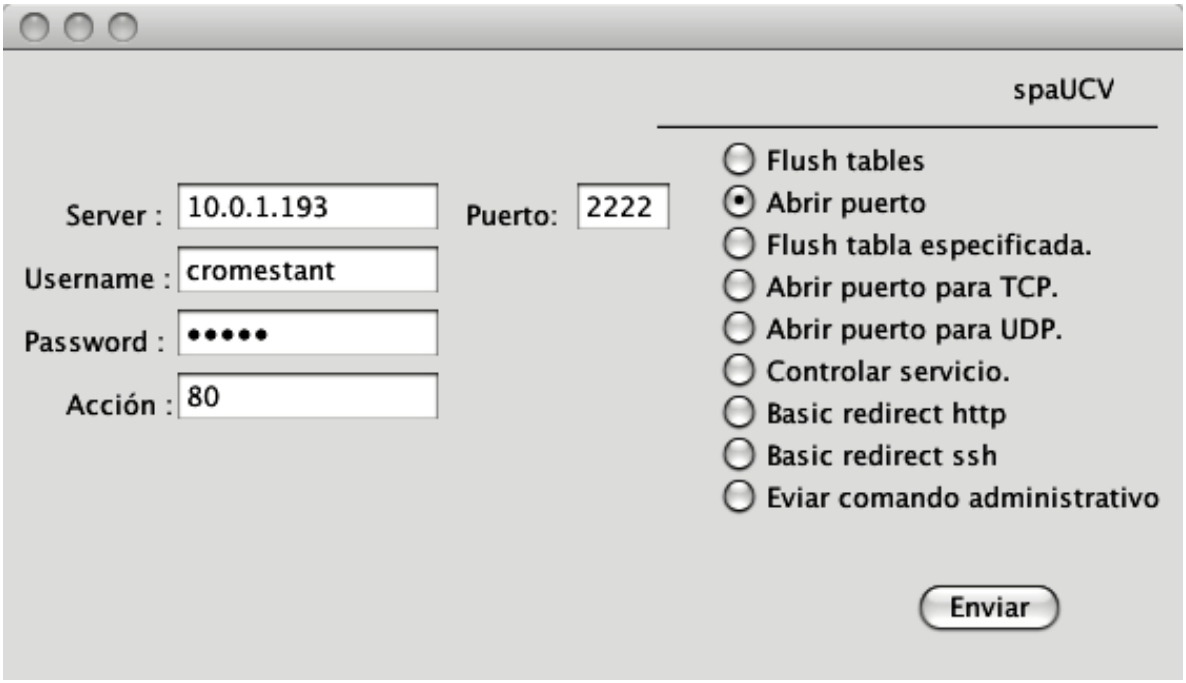

Fig. ## Acción 2 (abrir puerto 80).

Flush tabla específica:

Este acción permite vaciar las reglas de la tabla especificada en el campo acción del cliente en el firewall del servidor público.

Por ejemplo un comando ejecutado por el servicio de autenticación al recibir un paquete autenticado podria ser :

```
Case 5: Ejecutando iptables -F INPUT
```

Abrir un puerto para UDP:

Este acción , a diferencia del acción 2 , permite abrir un puertos (o un rango de puertos) UDP. El comando que se ejecuta en servidor es :

```
Case 3: Ejecutando iptables -A spa -p udp --dport 53 --source 10.0.1.199
-j ACCEPT
```

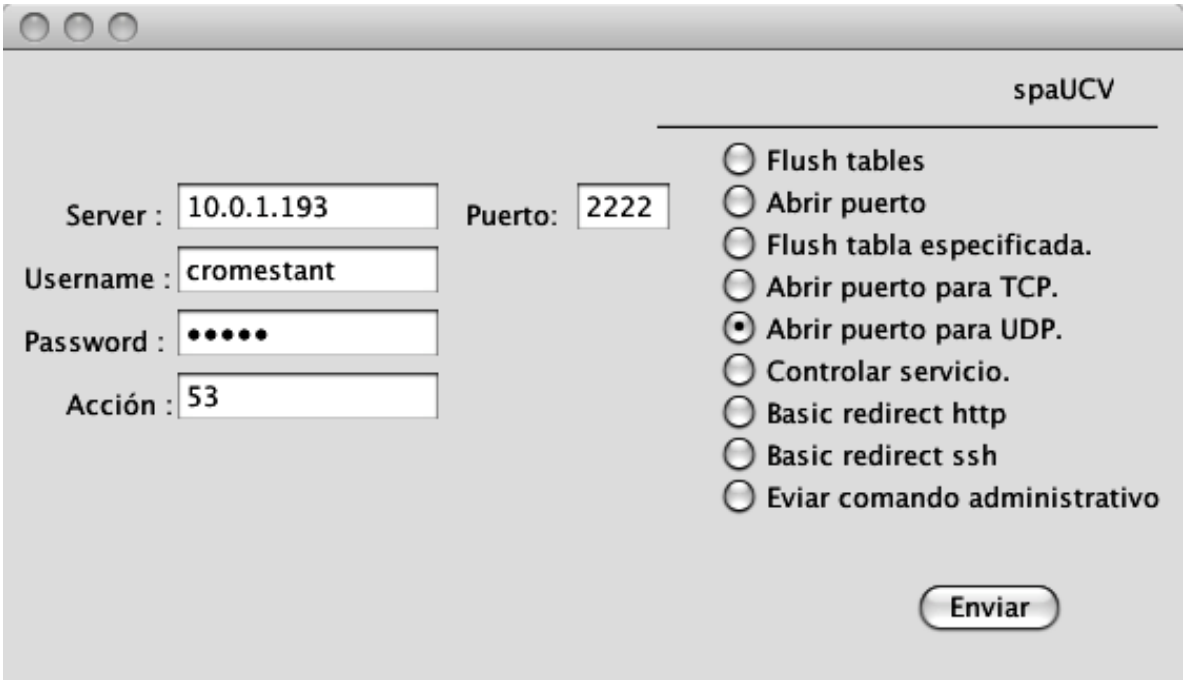

Fig. ## Abrir puerto UDP 53.

Controlar un servicio

Con esta acción el administrador del servidor puede tener control sobre los servicios que se están ejecutando en el servidor público. Tiene en sus manos, virtualmente una consola root en "/etc/init.d/" en donde puede controlar sus servicios como si estuviera administrando localmente.

Si por ejemplo el servicio de apache2 se estuviera funcionando mal, y se quisiera reiniciar el servicio para ver si esto solventa el problema, entonces el administrador pudiera hacerlo sin la necesidad de iniciar sesion en el servidor público. Un ejemplo de esto seria usar el campo acción con los valores `apache2 restart` (ver Fig. ##), y si el usuario esta permitido, entonces el servidor reiniciará el servicio.

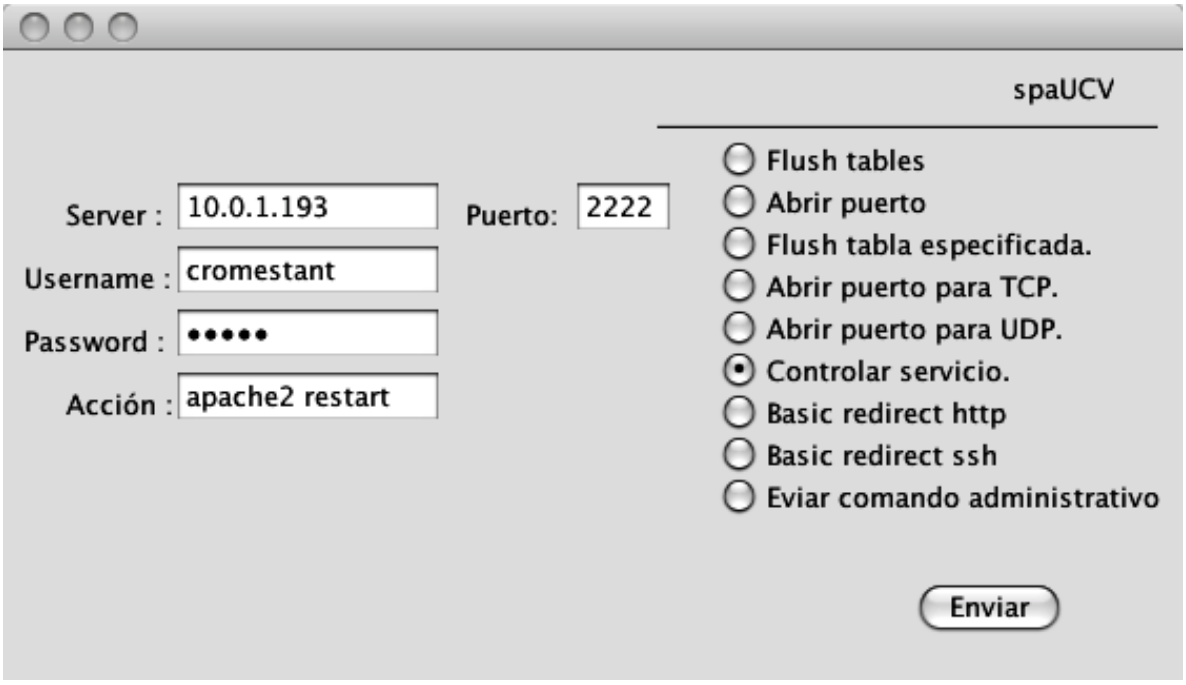

Fig.## Control de servicio apache2.

Basic redirects

El Basic Redirect es el nombre que se le da a las acciones que establecen traducción de direcciones de red para encaminar conexiones a otras maquinas, en nuestro ejemplo, “Basic redirect http” establece las reglas en el firewall para que las conexiones que vayan al puerto 80 del servidor público sean encaminadas hacia el servidor privado. De igual manera, el “basic redirect ssh”, hace lo mismo con las conexiones dirigidas al puerto 22. Esto permite iniciar conexiones desde afuera en un ambiente de conexión compartida por NAT (ref seminario).

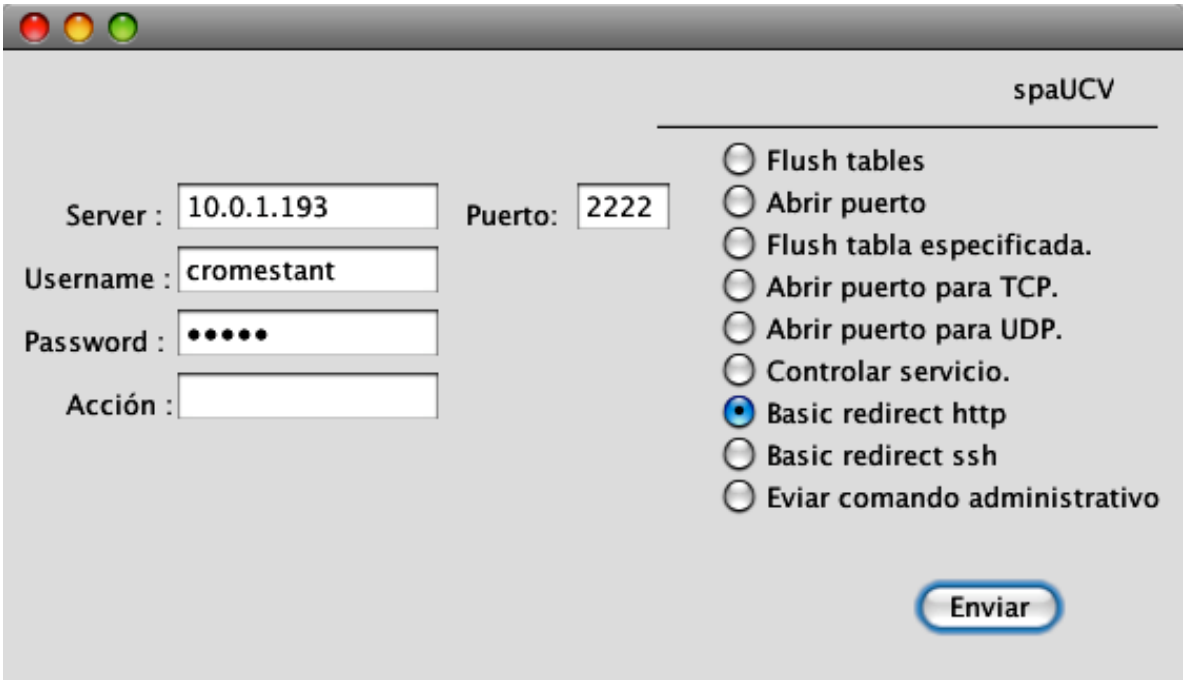

Fig. ## basic redirect http.

El servicio de autenticación establece las reglas necesarias para poder encaminar estas conexiones , un ejemplo de estas reglas seria:

```
Case 8 : BASIC CONFIG : Using Iptables Rule :

iptables -t nat -A PREROUTING -i wlan1 -s 10.0.1.199 -d 10.0.1.193 -p tcp
--dport 80 -j DNAT --to-destination 10.0.10.1

iptables -t filter -A FORWARD -i wlan1 -s 10.0.1.199 -o eth0 -d 10.0.10.1
-p tcp --dport 80 -j ACCEPT

iptables -t filter -A FORWARD -i eth0 -o wlan1 -s 10.0.10.1 -d 10.0.1.199
-p tcp --sport 80  -j ACCEPT

iptables -t nat -A POSTROUTING -o wlan1 -s 10.0.10.1 -d 10.0.1.199 -p tcp
--sport 80 -j SNAT --to-source 10.0.1.193
```

Enviar comando administrativo

Esta acción permite ejecutar cualquier comando en la consola del servidor público con permisos de root. Es evidente lo peligroso que puede ser esto, pero se incluyó como un ejemplo de lo que se puede hacer al configurar una acción en el

servidor, y para un usuario no administrador, esta opción puede ser deshabitada en su cliente. En esencia, el campo acción de la interfaz gráfica del cliente, se convierte en un terminal virtual, se pueden introducir cualquier comando bash que se desee en el, como por ejemplo la ejecución de un script (ver fig. ##).

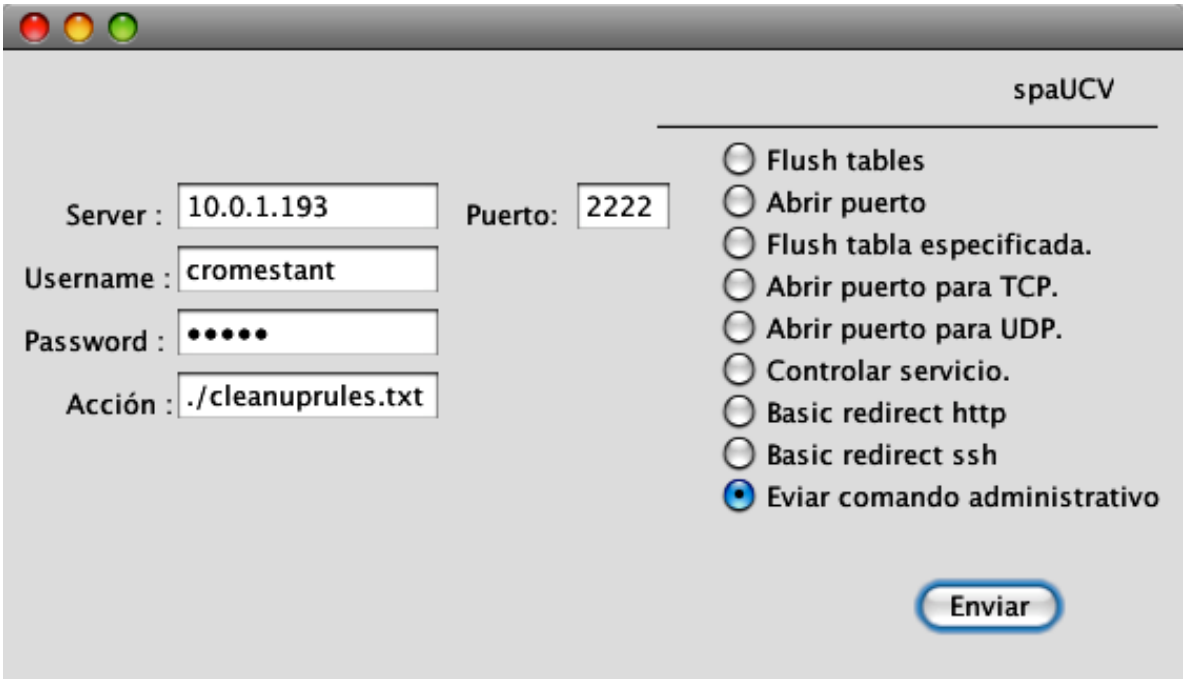

Fig. ## Comando administrativo.

Pruebas de rendimiento

El servicio de autenticación es un proceso que esta siempre a la espera de paquetes en la interfaz de red. Al recibir un paquete este debe procesarlo parcialmente antes de determinar si se ignora o si sigue analizándolo. Esto es una carga extra para el servidor, ya que aunque el firewall esta rechazando indiscriminadamente todos los paquetes, por detrás la aplicación está analizando cada uno , buscando alguno que tenga el formato adecuado, y solo en ese caso va a proceder a desencadenar alguna acción en el servidor.

Para reducir la carga de el servidor se integraron filtros , para limitar los paquetes que va a procesar la aplicación. Primero, solamente analiza los datagramas UDP (no se tocan los paquetes TCP), esto ya reduce la carga. Además se incluye un parámetro opcional (-p) para especificar el puerto UDP en el cuál se va a escuchar.

Con estos ajustes, el rendimiento del servidor no se ve afectado por el servicio (ver Fig.##).

Para luego verificar como reaccionaba el servidor al tener que trabajar con conexiones simultáneas se hicieron pruebas de conexión por parte de diez clientes simultáneos. Se pudo apreciar que esto tenia el mismo impacto que sin el servicio. Esto es, si las reglas del firewall están establecidas por defecto para rechazar las conexiones salvo que vengan de los clientes específicamente permitidos por las reglas. Es un hecho que mientras más reglas tengan las distintas tablas de iptables, más tardará en procesar el paquete el sistema, y por ello es lógico esperar que con una cantidad alta de clientes concurrentes, el sistema se vera afectado [Kadl-EK].

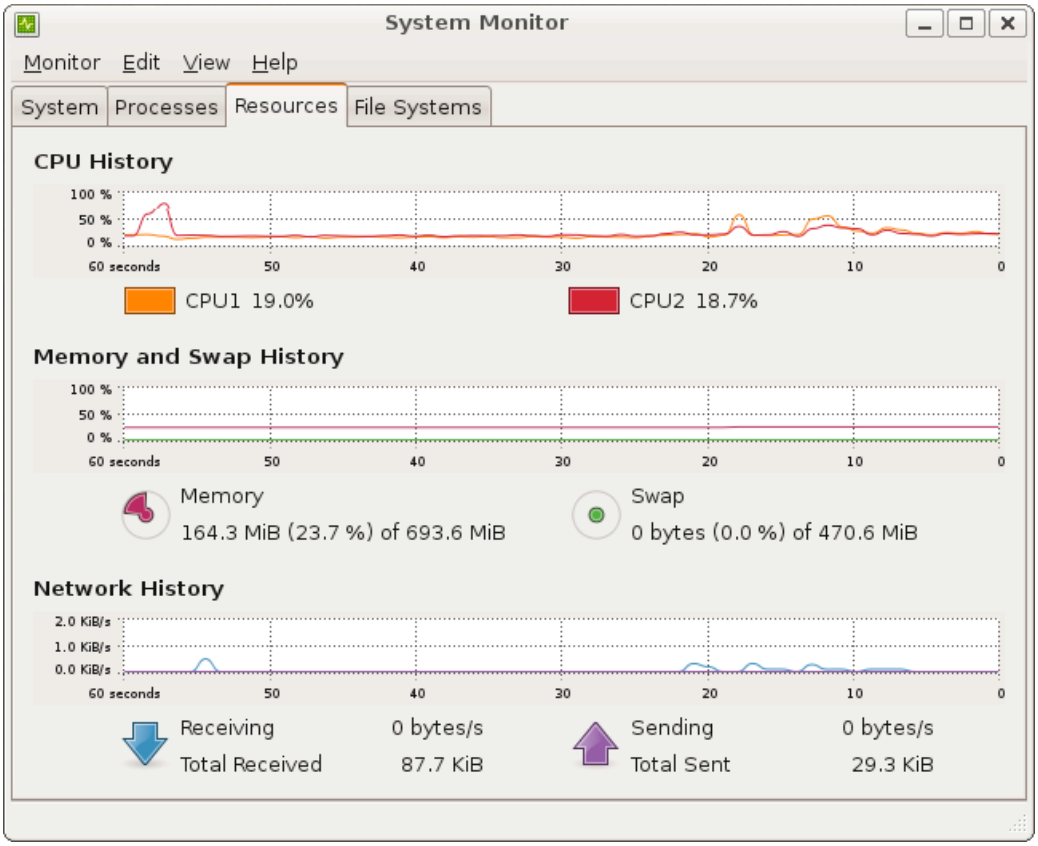

Fig.## Sistema con el servicio.

Pruebas de detección y barrido de puertos

A cada datagrama que se envia se le anexa en la carga útil una marca de tiempo, la cuál hace que aunque se envíe el mismo mensaje en dos ocasiones distintas, el mensaje no será el mismo.

Todas las comunicaciones entre el cliente y el servidor público están encriptadas con el algoritmo Blowfish [Schnei] , con una clave de 128 bits. Blowfish es un algoritmo de encriptado simétrico muy veloz, esto implica que la clave debe haber sido compartida previamente. Para efectos de este trabajo, la clave esta precompilada en el cliente y en el servicio de autenticación.

La ventaja de trabajar con datagramas encriptados y variantes (por el timestamp) es que vuelve un posible cripto-analisis sobre tramas capturadas mas complicado.

Por medio de wireshark [WSRK] se realizaron capturas de los paquetes que viajaban desde el cliente al servidor público.

```
20:27:44,447,565   ETHER

|0|00|11|50|10|72|fe|00|16|cb|bb|1b|07|08|00|45|00|00|44|a1|88|00|00|40|
11|c1|99|0a|00|01|c7|0a|00|01|c1|c6|11|08|ae|00|30|0b|b2|26|2f|bd|b1|50|
a3|7d|ae|39|aa|93|fc|54|34|73|aa|79|cc|12|4f|73|b5|91|f9|fa|43|34|c2|77|
2d|a5|b9|a8|67|68|ba|e9|c9|ed|38|

+---------+---------------+----------+

20:28:30,036,335   ETHER

|0|00|11|50|10|72|fe|00|16|cb|bb|1b|07|08|00|45|00|00|44|7a|66|00|00|40|
11|e8|bb|0a|00|01|c7|0a|00|01|c1|c6|0e|08|ae|00|30|dd|5a|26|2f|bd|b1|50|
a3|7d|ae|39|aa|93|fc|54|34|73|aa|79|cc|12|4f|73|b5|91|f9|54|83|3f|ff|08|
e4|31|bb|82|e3|82|01|58|b5|35|b0|
```

Fig. ## Captura de tramas.

Barrido de puertos

Utilizando nmap [Nmp] se realizaron barridos de puerto y deteccion de sistema operativo al sistema en los tres estados posibles : desprotegido, protegido, protegido abierto para un cliente.

Los resultados se presentan a continuación :

Sistema desprotegido:

El sistema tal cual es cuando no hay reglas estrictas de firewall. Este barrido revela mucha información que podría ser utilizada por un atacante.

```
Starting Nmap 4.68 ( http://nmap.org ) at 2008-08-17 16:28 SA Western
Standard Time

Initiating ARP Ping Scan at 16:28

Scanning 10.0.1.193 [1 port]

Completed ARP Ping Scan at 16:28, 0.16s elapsed (1 total hosts)

Initiating Parallel DNS resolution of 1 host. at 16:28

Completed Parallel DNS resolution of 1 host. at 16:28, 0.00s elapsed

Initiating SYN Stealth Scan at 16:28

Scanning 10.0.1.193 [1715 ports]

Discovered open port 80/tcp on 10.0.1.193

Discovered open port 22/tcp on 10.0.1.193

Completed SYN Stealth Scan at 16:28, 2.81s elapsed (1715 total ports)

Initiating Service scan at 16:28

Scanning 2 services on 10.0.1.193

Completed Service scan at 16:28, 6.01s elapsed (2 services on 1 host)

Initiating OS detection (try #1) against 10.0.1.193

SCRIPT ENGINE: Initiating script scanning.

Initiating SCRIPT ENGINE at 16:28

Completed SCRIPT ENGINE at 16:28, 0.03s elapsed

Host 10.0.1.193 appears to be up ... good.

Interesting ports on 10.0.1.193:

Not shown: 1713 closed ports
```

```
PORT   STATE SERVICE VERSION

22/tcp open  ssh     OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)

80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.3
with Suhosin-Patch)

|_ HTML title: Site doesn't have a title.

MAC Address: 00:11:50:10:72:FE (Belkin)

Device type: general purpose

Running: Linux 2.6.X

OS details: Linux 2.6.13 - 2.6.24

Uptime: 0.222 days (since Sun Aug 17 11:09:03 2008)

Network Distance: 1 hop

TCP Sequence Prediction: Difficulty=207 (Good luck!)

IP ID Sequence Generation: All zeros

Service Info: OS: Linux


Read data files from: C:\Program Files\Nmap

OS and Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 12.625 seconds

           Raw packets sent: 1739 (78.128KB) | Rcvd: 1732 (69.692KB)
```

Fig.## Barrido al sistema abierto.

Se puede verificar que el sistema abierto revela mucha información sobre el sistema, incluyendo los puertos abiertos, sistema operativo, servicios que corren en esos puertos y hasta versión de los servicios.

El objetivo de este sistema es proveer una capa extra de seguridad al sistema existente, para poder ocultar lo más posible el servidor manteniendo la conectividad.

El proximo barrido de puertos que se realizó fue con el sistema SPA funcionando, los resultados del barrido se presentan a continuación :

```
Starting Nmap 4.68 ( http://nmap.org ) at 2008-08-17 17:49 SA Western
Standard Time

Initiating ARP Ping Scan at 17:49

Scanning 10.0.1.193 [1 port]

Completed ARP Ping Scan at 17:49, 0.08s elapsed (1 total hosts)

Initiating Parallel DNS resolution of 1 host. at 17:49

Completed Parallel DNS resolution of 1 host. at 17:49, 0.00s elapsed

Initiating SYN Stealth Scan at 17:49

Scanning 10.0.1.193 [1715 ports]

Completed SYN Stealth Scan at 17:49, 38.92s elapsed (1715 total ports)

Initiating Service scan at 17:49

Initiating OS detection (try #1) against 10.0.1.193

Retrying OS detection (try #2) against 10.0.1.193

SCRIPT ENGINE: Initiating script scanning.

Host 10.0.1.193 appears to be up ... good.

All 1715 scanned ports on 10.0.1.193 are filtered

MAC Address: 00:11:50:10:72:FE (Belkin)

Too many fingerprints match this host to give specific OS details

Network Distance: 1 hop


Read data files from: C:\Program Files\Nmap

OS and Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 42.407 seconds

           Raw packets sent: 3479 (157.634KB) | Rcvd: 1 (42B)
```

Fig. ## Barrido al sistema con SPA.

Lo primero que se nota en estos resultados, es que el barrido tarda mucho mas tiempo (42 segundos en vez de los 12 en las pruebas anteriores). La única información que puede averiguar nmap sobre el host, es el fabricante de la interfaz de red , y esto es debido a la dirección MAC que se obtiene en las capas inferiores

a IP. A diferencia de la prueba anterior nmap no es capaz de detectar servicios ni versiones ni el sistema operativo. Esto es lo que se estaba buscando , ya que de esta manera, un atacante tiene más problemas para conseguir un vector de ataque al sistema.

Es de interés notar que en este modo, el sistema no responde a los mensajes ICMP, esto implica que un barrido horizontal simple no detectaría la presencia de un host en esa dirección IP; esto provee una capa de protección contra los *0 day exploits* .

Como última prueba, se realizaron barridos al servidor cuando este estaba atendiendo las peticiones de un host ya autenticado desde el host autenticado y desde otro host. Se utilizo la acción de abrir el puerto 80 (httpd).

Deste el host no autenticado, el resultado fue el mismo que en el caso anterior, nmap no pudo detectar nada más que el fabricante de la interfaz de red.

```
Starting Nmap 4.68 ( http://nmap.org ) at 2008-08-17 17:47 VET
Initiating ARP Ping Scan at 17:47
Scanning 10.0.1.193 [1 port]
Completed ARP Ping Scan at 17:47, 0.04s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 17:47
Completed Parallel DNS resolution of 1 host. at 17:47, 0.00s elapsed
Initiating SYN Stealth Scan at 17:47
Scanning 10.0.1.193 [1715 ports]
Discovered open port 80/tcp on 10.0.1.193
Completed SYN Stealth Scan at 17:48, 8.93s elapsed (1715 total ports)
Initiating Service scan at 17:48
Scanning 1 service on 10.0.1.193
Completed Service scan at 17:48, 6.03s elapsed (1 service on 1 host)
Initiating OS detection (try #1) against 10.0.1.193
SCRIPT ENGINE: Initiating script scanning.
Initiating SCRIPT ENGINE at 17:48
```

```
Completed SCRIPT ENGINE at 17:48, 0.02s elapsed

Host 10.0.1.193 appears to be up ... good.

Interesting ports on 10.0.1.193:

Not shown: 1714 filtered ports

PORT   STATE SERVICE VERSION

80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.3
with Suhosin-Patch)

|_ HTML title: Site doesn't have a title.

MAC Address: 00:11:50:10:72:FE (Belkin)

Warning: OSScan results may be unreliable because we could not find at
least 1 open and 1 closed port

Device type: general purpose

Running: Linux 2.6.X

OS details: Linux 2.6.13 - 2.6.24

Uptime: 0.278 days (since Sun Aug 17 11:08:15 2008)

Network Distance: 1 hop

TCP Sequence Prediction: Difficulty=202 (Good luck!)

IP ID Sequence Generation: All zeros


Read data files from: /usr/local/share/nmap

OS and Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 18.706 seconds

           Raw packets sent: 3468 (155.062KB) | Rcvd: 13 (690B)
```

Fig.## Barrido de puertos desde un host autenticado.

Desde el host autenticado, nmap pudo determinar cual servicio estaba corriendo y cual versión. Pero no pudo hacer mas que intentar determinar el sistema operativo (`OS details: Linux 2.6.13 - 2.6.24`), pero él mismo no considera que el resultado sea confiable, ya que no pudo encontrar suficiente información para determinar con seguridad el sistema operativo.

Se puede entonces ver que el sistema con SPA es más seguro que sin el, ya que presenta menos información al ambiente, dificultando así el ataque al sistema. Además,solamente la protección que provee a los barridos horizontales ya impide la mayoría de los ataques automatizados a los cuales son expuestos los sistemas a diario.

Pruebas de resistencia a la denegación de servicio

Una de las ventajas que brinda un sistema de autenticación sigilosa, es una protección parcial a los ataques de denegación de servicio (DoS). Para realizar esta pruebas se hicieron ataques al sistema abierto y luego al sistema protegido, intentando simultáneamente establecer una conexión legítima.

Para realizar los ataque se utilizaron dos herramientas , la primera es un simple , pero efectivo, script de perl que simplemente realiza miles de conexiones simultáneas al servidor, y las deja abiertas, dejando así saturado al servidor atendiendo peticiones. En el servidor se monitoreáron los recursos del sistema para ver como era afectado en general. La segunda herramienta que se utilizo es una herramienta de prueba de stress para servidores, desarrollada por la fundación apache: JMeter. Los resultados de estas pruebas se presentan a continuación:

Al inicar el ataque sobre el servidor, el sistema comenzo inmediatamente a utilizar mas recursos de CPU e interfaz de red (ver Fig. ##). Desde el lado del cliente, esto resultó efectivamente en una denegación de servicio, ya que no se pudo cargar una página web servida por el webserver instalado en apache durante todo el ataque. Esto puso en evidencia la vulnerabilidad que tiene el sistema cuando no está protegido.

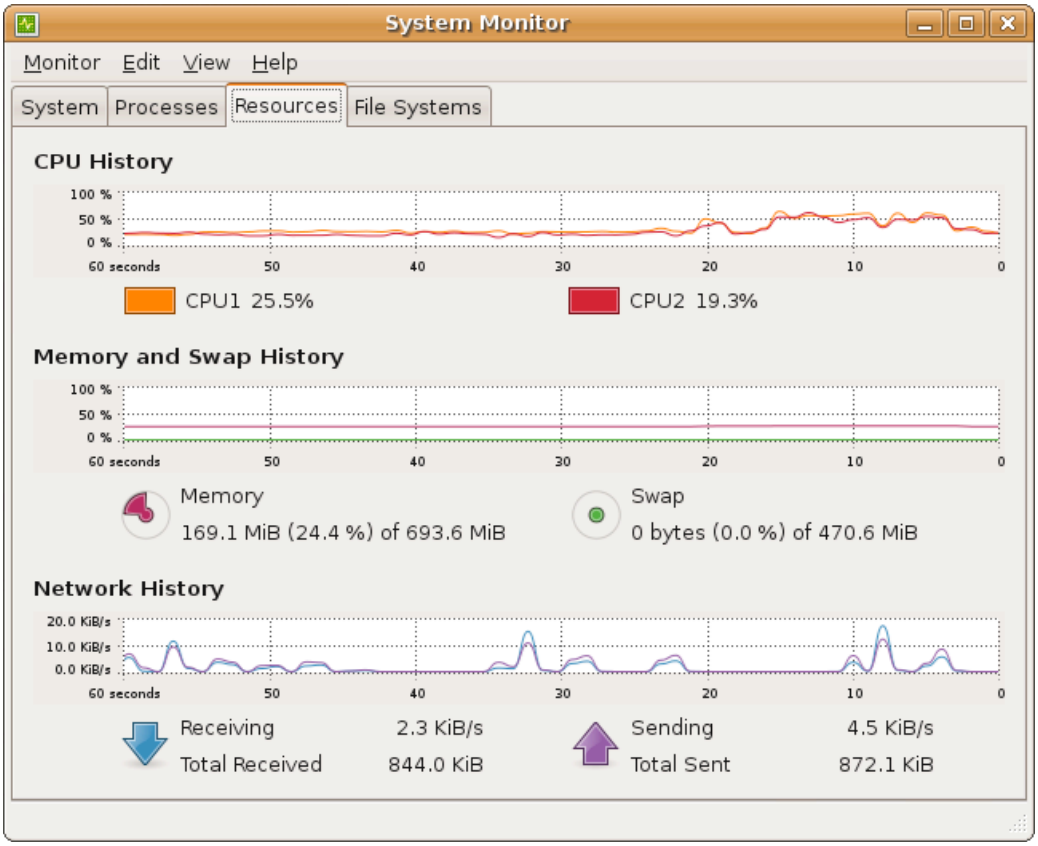

Fig. ## Ataque con script al servidor desprotegido.

Luego de haber realizado el ataque sobre el sistema desprotegido, se repitió el experimento sobre el servidor protegido por el sistema SPA. Se pudo apreciar que el rendimiento del sistema no era afectado por el ataque, existía una leve alza del uso de los recursos de CPU con respecto al servidor cuando no está siendo atacado (ver Fig. ## y Fig. ##).

Luego de haberse autenticado con el servicio de autenticación, el cliente fue capaz de utilizar los recursos del sistema. El ataque de denegación de servicio no fue exitoso.

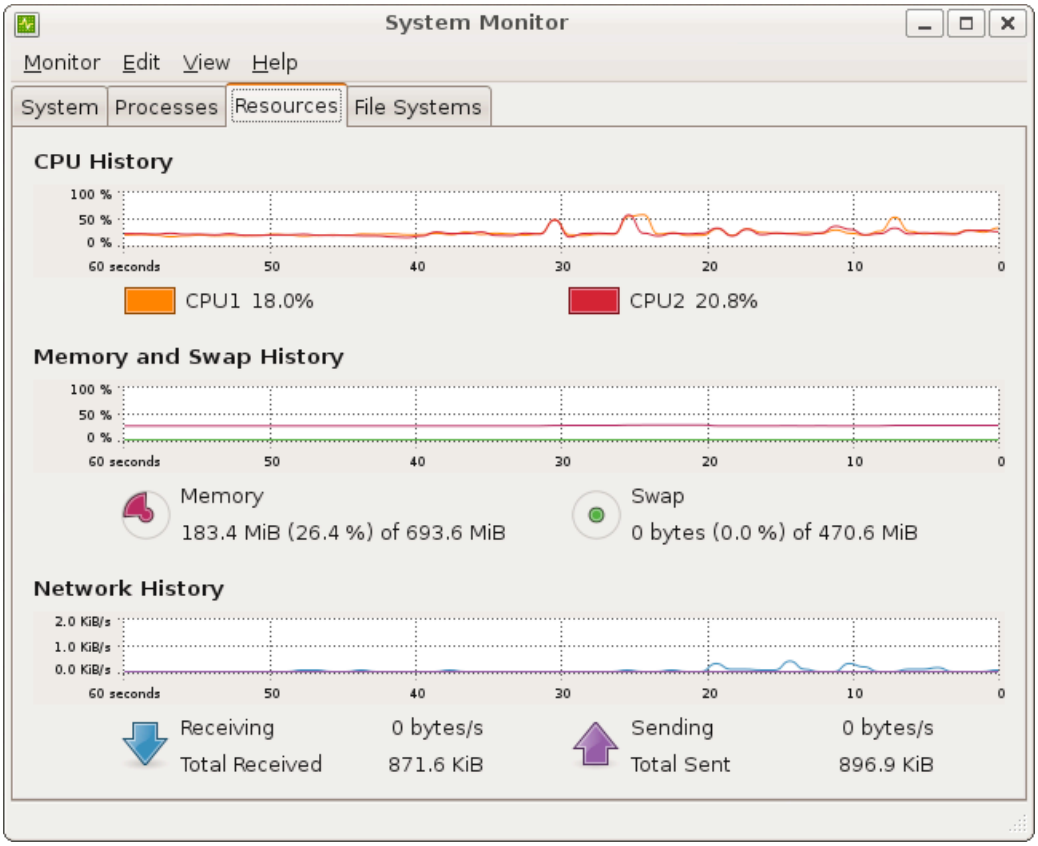

Fig ##. Sistema con SPA siendo atacado con el script.

Como últimas pruebas al sistema, se realizaron con JMeter pruebas de estrés al servidor web, con el sistema abierto y el cerrado. Los resultados fueron muy similares a los vistos con la prueba anterior. Con la ayuda de JMeter se simularon 500 conexiones simultáneas por segundo al servidor apache, esto a diferencia del script, son conexiones completas, se abren y cierran, así no se consumen recursos por conexiones abiertas no terminadas. En el sistema abierto se obtuvo una degradación del servicio con tiempos de respuesta muy altos (Ver Fig.##).

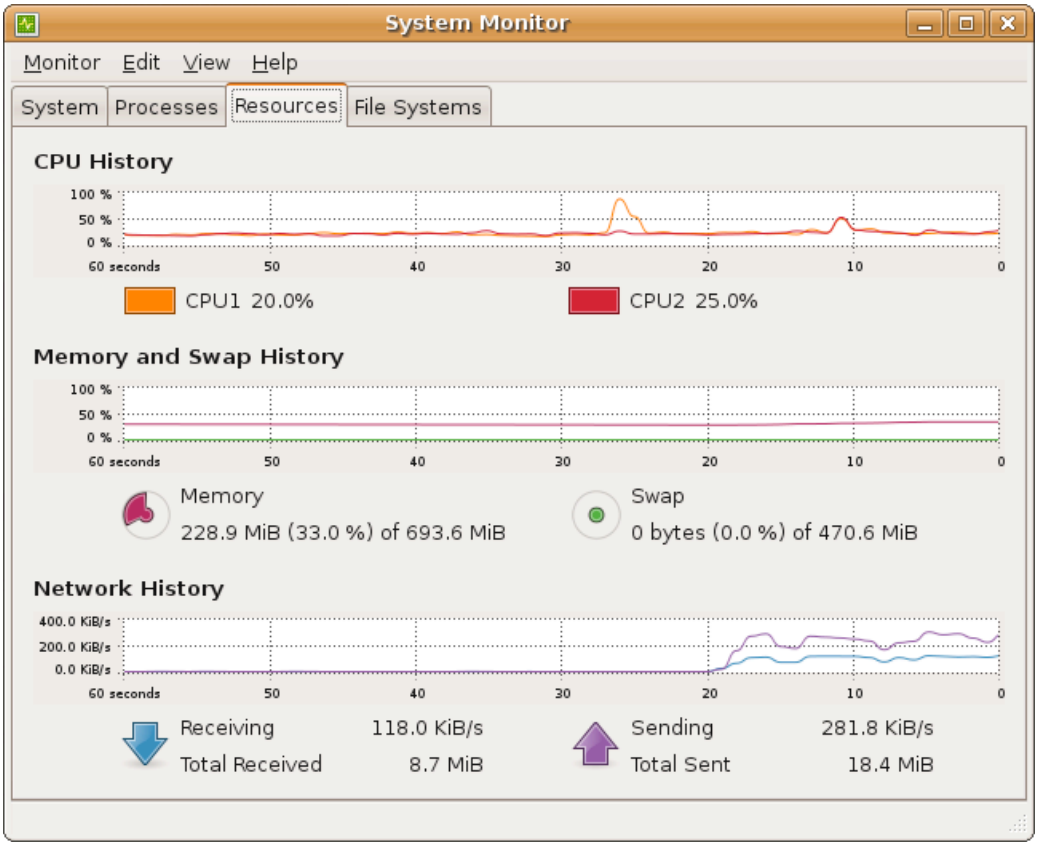

Fig.## Pruebas de estrés con JMeter al sistema abierto.

Al repetir la prueba de estrés con el servidor protegido por el sistema, se pudo apreciar que no se degradaba en nada el funcionamiento del sistema (ver Fig.##), las conexiones realizadas durante la prueba de estrés fueron todas exitosas, y los tiempos de respuesta eran instantáneos.

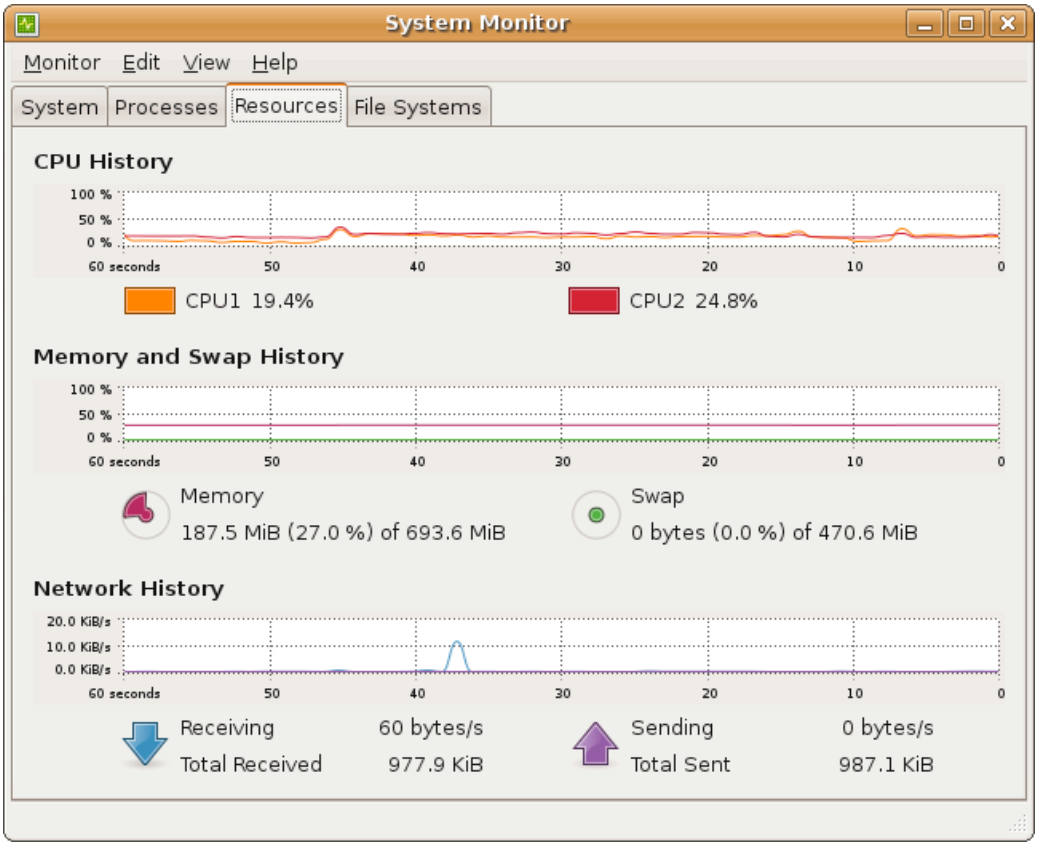

Fig.## Pruebas de estrés con el sistema protegido.

Conclusion y recomendaciones

Al terminar este trabajo se tiene que los sistemas de autenticación sigilosa no solo funcionan, sino que también pueden ser extendidos para realizar cualquier acción en el servidor que se desee. Se cumplió con el requerimiento de desarrollar y probar un sistema de autenticación sigilosa como capa de seguridad sobre un sistema existente.

Estos sistemas funcionan desde el punto de vista de incremento de seguridad ya que con las reglas estrictas del firewall se pueden esconder al naturaleza del host que reside allí , aunque no se puede esconder la presencia ya que esto se resuelve por medio del protocolo ARP en capas inferiores.

Los ataques automatizados que se basan en barridos horizontales , los cuales tienen por propósito encontrar hosts que tienen un servicio vulnerable , se vuelven completamente inútiles frente a sistemas protegidos por un esquema de autenticación sigilosa. De esto se deriva una protección muy efectiva contra el malware, ya que se brinda una capa de protección de servicios, y la identificación de servicios vulnerables es dificultada.

Las reglas del firewall en el sistema permiten descartar paquetes no autorizados, lo que permite que se ahorren recursos del sistema en procesar paquetes que luego serian descartados. Esto brinda una protección contra ciertos ataques de denegación de servicios , los cuales saturan el pool de conexiones disponibles en el equipo: No se establece conexión si no se ha autenticado antes.

Los problemas que se observaban con los esquemas estrictos de firewall son solucionados ahora con la implementación de reglas dinámicamente establecidas. Se autoriza basándose en las credenciales del usuario y el cliente que utiliza para autenticarse y no en su localización. Además con el sistema de limpieza de conexiones viejas, no quedan en el firewall reglas viejas que pudieran ser un problema de seguridad para el sistema. A lo sumo, una regla queda huérfana por cinco minutos (este valor puede ser modificado en el script que utiliza el crontab para limpiar las conexiones).

Las comunicaciones de autenticación son encriptadas con un esquema de criptografía considerado seguro y como se incluye en cada comunicación una marca de tiempo, se dificulta el criptoanálisis de las comunicaciones capturadas. Esto protege las credenciales del usuario y brinda protección a las falsas autenticaciones: para autenticarse no solo hay que tener las credenciales adecuadas, sino que hay que tener también la clave criptográfica utilizada en el sistema.

Esta implementación brinda mucha flexibilidad para el administrador del sistema; le permite tener acceso a una consola administrativa remota, sin tener que iniciar sesión en el sistema. Además se provee la acción “Basic redirect” la cual permite establecer reglas de NAT dinámicas, facilitando su trabajo como administrador del firewall; ahora se puede abrir una conexión desde afuera en un ambiente de NAT.

La implementación en C y la utilización de librerías de bajo nivel para la captura de paquetes hacen de este sistema una capa de protección muy eficiente; el servidor no se ve afectado por la nueva carga que se le adiciona. Más aún, si el servidor es atacado frecuentemente, el sistema llega a reducir su carga.

Se recomienda continuar a estudiar este tipo de sistemas ya que pueden llegar a ser herramientas muy útiles y accesibles para las empresas que no disponen de los recursos para establecer una seguridad de alto nivel en sus servidores. Esta implementación sirve de referencia académica sobre todo en el campo de seguridad en redes.

Seria provechoso para el campo si se realizara una implementación de un sistema de autenticación sigilosa que aprovechase el esquema de autenticación utilizado por el servidor (sea el shadow, ldap, kerberos) en vez de depender de una lista de usuarios en servidor. Más aún si se implementara con las permisologías establecidas ya por las listas de acceso en servidor , ya que en la implementación presente, todos los usuarios gozan de los mismos permisos: root.

Anexo 1:

Codigo de encriptado/desencriptado:

Encriptado con bouncy-castle:

```
package spaclient;

/**
 *
 * @author cromestant
 */
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.bouncycastle.util.encoders.Hex;

public class CriptoEngine {
private IvParameterSpec salt;
private SecretKey skeySpec;
private Cipher c;
    public CriptoEngine(byte[] iv, byte[] key){
        try{
        salt = new IvParameterSpec(iv);
        skeySpec = new SecretKeySpec(key, "Blowfish");
        c = Cipher.getInstance("Blowfish/CBC/PKCS5Padding", "BC");
        c.init(Cipher.ENCRYPT_MODE, skeySpec, salt);
        }catch(Exception e) {
      e.printStackTrace();
}
        }

public byte[] encryptPayload(String payload) throws
IllegalBlockSizeException, BadPaddingException{
    byte hex[]=Hex.encode(payload.getBytes());
    System.out.println(hex.toString());
    return c.doFinal(hex);
}
public byte[] encryptPayload(byte[] payload) throws
IllegalBlockSizeException, BadPaddingException{
    //byte hex[]=Hex.encode(payload.getBytes());
    System.out.println(payload.toString());
    return c.doFinal(payload);
}

}
```

Desencriptado con openssl:

```
#include <openssl/evp.h>
#include <openssl/blowfish.h>
#include <openssl/crypto.h>
#include <openssl/bn.h>
#include <openssl/pem.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>


      EVP_CIPHER_CTX ctx;
int do_decrypt(unsigned char *output,unsigned char *input,int size)
{
      unsigned char *cripted;

      int criptedlen, tmplen,plainlen,errcode;
      long retsize;
      unsigned char key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
      unsigned char iv[] = {1,2,3,4,5,6,7,8};
      printf("entering decript routine.Size : %d",size);
      EVP_CIPHER_CTX_init(&ctx);
      printf("decript routine ctx.");
      EVP_DecryptInit_ex(&ctx, EVP_bf_cbc(), NULL, key, iv);
      if(!EVP_DecryptUpdate(&ctx, output , &plainlen,input,size))
              {
                /* Error */
                   printf("error 1");
              return -1 ;
               }

      printf("\nTamano bloques ppales : %d, \n",plainlen);
      if(!(errcode=EVP_DecryptFinal_ex(&ctx, output + plainlen,
&tmplen)))
            {
                /* Error */
                printf("error 2, code %x %s",errcode,output);
             return -1;
               }
        plainlen += tmplen;
      output[plainlen+1] ='\0';
      printf("\n\ntotal decrypt size : %d\n",plainlen);
      printf("decript : %s\n\n\n\n",output);
        EVP_CIPHER_CTX_cleanup(&ctx);

        return plainlen;
}
```

Codigo de captura de paquetes:

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const
u_char *packet)
{

      static int count = 1;                   /* packet counter */
      int flag =0;
      /* declare pointers to packet headers */
      const struct sniff_ethernet *ethernet;  /* The ethernet header [1]
*/
      const struct sniff_ip *ip;              /* The IP header */
      const struct sniff_tcp *tcp;            /* The TCP header */
      const struct sniff_udp *udp;
      const unsigned char *payload;                    /* Packet payload
*/
      unsigned char *input, *output;
      int decryptedSize;
      int size_ip;
      int size_tcp,size_udp;
      int size_payload;

      FILE *outf;
      printf("\nPacket number %d:\n", count);
      count++;

      /* define ethernet header */
      ethernet = (struct sniff_ethernet*)(packet);

      /* define/compute ip header offset */
      ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
      size_ip = IP_HL(ip)*4;
      if (size_ip < 20) {
            printf("   * Invalid IP header length: %u bytes\n", size_ip);
            return;
      }

      /* print source and destination IP addresses */
      //Save source and destination IP addresses then Print.
      if(!ipFlag)
      {
                  memcpy(currentIp,inet_ntoa(ip-
>ip_src),strlen(inet_ntoa(ip->ip_src)));
            flag=1;
      }
            memcpy(ownIp,inet_ntoa(ip->ip_dst),strlen(inet_ntoa(ip-
>ip_dst)));


      printf("       From: %s||%s\n", inet_ntoa(ip->ip_src),currentIp);
      //currentIp= inet_ntoa(ip->ip_src);
      printf("         To: %s||%s\n", inet_ntoa(ip->ip_dst),ownIp);
      //ownIp = inet_ntoa(ip->ip_dst);
      /* determine protocol */
      switch(ip->ip_p) {
            case IPPROTO_TCP:
```

```
                  printf("   Protocol: TCP\n");
                  break;
            case IPPROTO_UDP:
                  printf("   Protocol: UDP\n");
                  flag =1;
                  break;
            case IPPROTO_ICMP:
                  printf("   Protocol: ICMP\n");
                  return;
            case IPPROTO_IP:
                  printf("   Protocol: IP\n");
                  return;
            default:
                  printf("   Protocol: unknown\n");
                  return;
      }

/*
 * if flag, then UDP else TCP
 * */


      if (flag){
            /*Parsear el header UDP, computar el offset y distancia del
payload.*/

            udp = (struct sniff_udp*)(packet +SIZE_ETHERNET+size_ip);
            printf("   Src port: %d\n", ntohs(udp->uh_sport));
            printf("   Dst port: %d\n", ntohs(udp->uh_dport));
            payload = (u_char *)(packet + SIZE_ETHERNET + size_ip +8 );
            size_payload = ntohs(udp->udp_len)-8;
            printf("Payload size : %d.\n ",size_payload);
            printf("Payload 1 : \n%.*x\n",size_payload,payload);
      unsigned char tpayload[size_payload];
      memcpy(tpayload,payload,size_payload);

      output = (unsigned char *)malloc(sizeof(unsigned char
*)*size_payload +8);
      decryptedSize=do_decrypt(output,tpayload,size_payload);

            if (decryptedSize==-1){
                  printf("crap");
                  return ;
                  }
            printf("Payload 2: %s\n",output);
            printf("\n%s\n",lista->username);
            //procesar ahora el packete recibido a ver si tiene el
formato adecuado
            int retval =processAP(output,decryptedSize);
            if (retval == 1){
                  processAction();
            }
            free(output);
      }else{
            /* define/compute tcp header offset */
            tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
```

```
            size_tcp = TH_OFF(tcp)*4;
            if (size_tcp < 20) {
                  printf("   * Invalid TCP header length: %u bytes\n",
size_tcp);
                  return;
            }

            printf("   Src port: %d\n", ntohs(tcp->th_sport));
            printf("   Dst port: %d\n", ntohs(tcp->th_dport));

      /* define/compute tcp payload (segment) offset */
            payload = (u_char *)(packet + SIZE_ETHERNET + size_ip +
size_tcp);

      /* compute tcp payload (segment) size */
            size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);

      /*
       * Print payload data; it might be binary, so don't just
       * treat it as a string.
       */
            if (size_payload > 0) {
                  printf("   Payload (%d bytes):\n", size_payload);
                  //print_payload(payload, size_payload);
                  printf("Payload :\n %x\n",payload);
            }
      }
      return;
}
```

CRON script:

```
#!/usr/bin/php -q
<?php
/*Este es el script de  verificacion de logs de SPA UCV
TEG de Charles Romestant, CI 14486880
requiere tener php5-cli instalado.
*/
$min =5;
$timeOut =$min*60;
$dir ="/var/spaUCV/";
if ($handle = opendir($dir))
{

      $now = time();
      while (false !==($file =readdir($handle)))
      {
            if (($file!==".") and ($file!==..") and ($file!
='spaCron.php'))
            {
                  $fh = fopen($dir.$file,"r");
                  $ts=fgets($fh);
                  if (($now-$ts)>$timeOut)
                  {
                        while($action = fgets($fh)){
                        exec($action);
                        }
                        fclose($fh);
                        unlink($dir.$file);
                  }else
                  {
                        fclose($fh);
                  }
            }
      }
      closedir($handle);
}
?>
```

Anexo 2

Configuración servidor

Librerías criptográficas

Las librerías criptográficas utilizadas en este trabajo son las provistas por el paquete openssl.

La instalación de estas librerías en el sistema operativo Ubuntu se hace de la siguiente manera :

```
sudo aptitude install libssl0.9.8
```

Esto instala las cabeceras, las librerías de desarollo y las páginas “man”.

Estas librerías necesitan ser enlazadas al momento de compilar , utilizando las siguientes opciones de compilación :

```
-lssl -lcrypto
```

Por ejemplo :

```
gcc programa.c -o programa -lssl -lcrypto
```

Más información se puede encontrar en el sitio web de openssl (http://openssl.org).

Limpieza de conexiones

Repositorio de archivos “rollback”

En la aplicación del servidor de autenticación se define un directorio para que el aplicativo escriba sus bitácoras de comandos iptables a revertir. Se definió el directorio “/var/spaUCV” para este fin.

```
sudo mkdir /var/spaUCV
sudo chmod 700 /var/spaUCV
```

Esto crea el directorio y lo protege para que otros usuarios no autorizados puedan tener acceso a los archivos. Por comodidad, el script que maneja el reverso de las conexiones lo guardaremos también en este directorio, y se le otorgaran permisos de ejecución:

```
sudo cp spaUCV.php /var/spaUCV/.
sudo chmod +x /var/spaUCV/spaUCV.php
```

Luego, para que se ejecute periódicamente este debe ser agregado al *crontab* del usuario ROOT :

```
sudo crontab -e
*/5 * * * * /var/spaUCV/spaUCV.php >> /var/log/spaUCV.log
```

librerías de captura de paquetes :LIBPCAP

Para realizar la captura de paquetes por detrás del firewall, es necesario instalar las librerías de captura de paquetes a bajo nivel de PCAP, estas permiten capturar a nivel de kernel los paquetes que llegan a la interfaz de red. Por esto , el programa que desee utilizarlas, debe ser ejecutado con permisología ROOT.

Para instalar estas librerías en UBUNTU se debe ejecutar el siguiente comando :

```
sudo aptitude install libpcap0.7-dev
```

Estas librerías también deben ser enlazadas en momento de compilación, de la siguiente manera :

```
gcc programa.c -o programa -lpcap
```

makefile para el proyecto:

Para compilar el proyecto se puede usar el siguiente makefile con “make all”:

```
OBJS = spa.o decrypt.o myhandle.o
CC = gcc
DEBUG= -g
LFLAGS = -lpcap -o
CRYPTF = -lssl -lcrypto
CFLAGS = -c  $(DEBUG)

decrypt.o :
	$(CC) $(CFLAGS) decrypt.c
myhandle.o:
	$(CC) $(CFLAGS) myhandle.c
spa : $(OBJS)
	$(CC) $(OBJS) $(LFLAGS) spa $(CRYPTF)
spa.o :
	$(CC) $(CFLAGS) spa.c
all : clean spa

clean:
	\rm *.o spa
```

Configuraciones del cliente

El cliente fue desarrollado de JAVA versión "1.5.0_13" , la interfaz gráfica fue desarrollada utilizando netbeans, por ello se requiere que las librerías gráficas de netbeans estén instaladas o empaquetadas con el .jar.

La librería criptográfica utilizada en el cliente es bouncycastle (http:// bouncycastle.org). Se escogió esta librería porque es una extensión para *Java Cryptography Extension (JCE)* y el *Java Cryptography Architecture (JCA).*

BouncyCastle provee una implementación apegada a las especificaciones oficiales de Blowfish [blowfish site], lo que hace que sea completamente compatible con la implementación de openssl. Esto permite encriptar en cliente java y luego desencriptar en el servidor.

Para poder acceder a las claves criptográficas de mas de 40 bits en Java se tiene que descargar del sitio de Java los "*unrestricted policy files"* los cuales permiten la utilización de claves criptográficas fuertes. Como la aplicación utiliza claves de 128 bits, es necesario hacer esta instalación. Estos archivos se pueden encontrar en la página de descargas de la máquina virtual y el kit de desarrollo de software.

Se tiene que descargar el JCE pluggin de bouncycastle y luego integrarlo al JDK de java que se tenga instalado. Esto tiene dos pasos en realidad:

1. Se tiene que copiar el `bcprov-jdk##.jar` path_java\jre\lib\ext y path_java/ \lib\ext.
2. Editar las políticas de seguridad del JDK y el JRE en : path_java\lib\security , hay que agregar una linea al final que diga "`security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider`".

Nota: security.-provider.-n , donde n es un número secuencial, en el ejemplo es 6 pero si hay otros proveedores instalados, este n puede ser otro número. Solo tiene que ser secuencial.

Glosario

Autenticación: Verificación y validación de identidad.

Bitácora: Registro de eventos del sistema, en el caso de este trabajo se trata de las reglas modificadas por cada conexión exitosa.

Confidencialidad: Servicio criptográfico que garantiza que solamente entes autorizados puedan tener acceso al contenido.

Cron y crontab: Del griego cronos , es el nombre de el planificador de tareas de los sistemas linux. Permite programar tareas por unidad de tiempo.

DoS: Denial of Service, denegación de servicio. No se puede tener acceso a los servicios brindados por un host, generalmente por culpa de un ataque malicioso al sistema.

Firewall: es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red que haya definido la organización responsable de la red.

ICMP: Parte integral del protocolo de Internet (IP) que resuelve errores y controla los mensajes.

Integridad: Servicio criptográfico que garantiza que el mensaje recibido es una copia exacta del mensaje enviado.

IP: Internet Protocol, protocolo estándar que define a los datagramas IP como la unidad de información que pasa a través de una red de redes y proporciona las bases para el servicio de entrega de paquetes sin conexión y con el mejor esfuerzo.

Librería: Nombre que se le da a una colección de rutinas con fin común distribuidas en un paquete (por ejemplo libc, la librería estándar de c). Se debe incluir su cabecera en un programa si se desea usarla.

Log: Ver bitácora.

NAT: Network Address Translator, sistema que se encarga de convertir direcciones privadas en direcciones públicas para lograr compartir una dirección pública (o más) entre varios host.

Port-knocking: Método de autenticación sigilosa que codifica la clave secreta en la secuencia de puertos a los cuales se envia un paquete.

Protocolo: Descripción formal de formatos de mensajes y reglas que dos o más máquinas deben seguir para intercambiar mensajes. Los protocolos pueden

describir detalles de bajo nivel de las interfaces de máquina a máquina o del intercambio entre programas de aplicación.

Rollback: Nombre que se le da a la acción que retrocede cambios en un sistema.
Router/Enrutador: dispositivo dedicado, de propósito especial, que se conecta a dos o más redes y envía paquetes de una red a otra. En particular, un ruteador IP envía datagramas IP entre las redes a las que está conectado.

SPA: Método de autenticación sigilos derivado del port-knocking. Se codifica la clave en la carga útil de un datagrama, aumentando las posibilidades y reduciendo las posibles fallas del método.

TCP: Transport control Protocol, protocolo de capa de transporte estándar que proporciona el servicio de flujo confiable full duplex y del cual dependen muchas aplicaciones.

Statefull: Con capacidad de mantener un estado.

Stateless: Sin capacidad de mantener un estado.
UDP: User Datagram Protocol, protocolo estándar que permite a un programa de aplicación enviar un datagrama hacia el programa de aplicación en otra máquina. El UDP utiliza el IP ara entregar datagramas. Conceptualmente la diferencia entre los datagramas UDP y los IP es que el UDP incluye un numero de puerto de protocolo, lo que permite al emisor distinguir entre varios programas de aplicación en una máquina remota dada. En la practica UDP también incluye una suma de verificación (checksum) opcional en el datagrama que esta enviando.

Referencias

[COMER96] *Douglas E. Comer – redes globales de información con Internet y TCP/IP (3ra edición)- 1996.*

[JEAN06] Sebastien Jeanquier , An Analysis of Port Knocking and Single Packet Authorization. Royal Holloway University of London, Inglaterra sept. 2006.

[ERIC03] *Jon Ericsson: HACKING -THE ART OF EXPLOITATION - 2003.*

[BELL94] *Steven M. Bellovin and William R. Cheswick. Network Firewalls. IEEE Communications Magazine, 32(9):50–57, September 1994.*

[FAR-VEN06] *Dan Farmer, Wietse Venema : Forensic Discovery- Preface- 2006.*

[DeGraaf07] *Reinderd G. deGraaf- Enhancing Firewalls: Conveying User and Application Identification to Network Firewall. Universidad de Calgary, Canada Mayo 2007.*

[KAP08] Kapersky labs article - *http://www.kaspersky.com/reading_room?chapter=207716701%5D*

[Khak-Chao07] *Amir R. Khakpour, and Hakima Chaouchi,* ***ESSTCP: Enhanced Spread-Spectrum TCP*** *in Proc. of the 3rd International Workshop on Security in Systems and Networks (SSN'07) in conjunction with IPDPS 2007, Long Beach, CA, March, 2007.*

[Schnei] *Bruce Schneider - Blowfish algorithm. http://www.schneier.com/blowfish.html*

[Kadl-EK] *József Kadlecsik, György Pásztor, Netfilter Performance Testing http://people.netfilter.org/kadlec/nftest.pdf*

Enlaces

[BC] http://www.bouncycastle.org

[DEB-BENCH]- *Debian made languaje benchmarks. http://shootout.alioth.debian.org/*

[WSRK] - http://www.wireshark.org/

[Nmp] - http://insecure.org