UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica

UNA INTERFAZ PARA LA IMPLEMENTACIÓN DE CONTROLES DE ACCESO OBLIGATORIO EN UNA ARQUITECTURA CLIENTE-SERVIDOR

TESIS

Que para obtener el grado de:

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta:

Bricia Elena Corona Pacheco

Dr. Juan Manuel García García

Director de tesis

Abril 2010

Resumen

En este trabajo de investigación presentamos el desarrollo de una interfaz de software, la cual hemos nombrado CS-Enhanced. Esta interfaz tiene como finalidad la implementación de un control de acceso obligatorio en una arquitectura Cliente-Servidor. La implementación del control de acceso se hace por medio de la creación de un conjunto de archivos de bajo nivel en SELinux e IPtables. Definimos un lenguaje de alto nivel para especificar políticas de control de acceso en una arquitectura Cliente-Servidor.

Desarrollamos un compilador que traduce del lenguaje de alto nivel a especificaciones de bajo nivel en SELinux e IPtables. Esta herramienta permite que la configuración del control de acceso obligatorio para una aplicación Cliente-Servidor se haga de una manera sencilla y correcta, inclusive si el usuario no cuenta con conocimientos previos sobre SELinux e IPtables.

El resultado que se obtiene con el uso de la herramienta es el directo mejoramiento de la seguridad sobre la aplicación Cliente-Servidor a la que se aplica, además que la facilidad de uso que CS-Enhanced proporciona al usuario la convierte en un herramienta de fácil adopción.

Abstract

In this research work, we present the development of a software interface, which we named CS-Enhanced. The goal of this interface is the implementation of a mandatory access control on a Client-Server architecture. The access control implementation is made by the creation of a set of low level files in SELinux and IPtables. We define a high level language to specify control access policies on a Client-Server architecture.

We develop a compiler that translates from the high level language to low level specifications in SELinux and IPtables. This tool allows an easy and correct configuration of mandatory access control for a Client-Server architecture, even if the sysadmin don´t know anything about SElinux and IPtables.

The result in the use of this tool is the improvement of the security of the application Client-Server in which is used, furthermore the simplicity in the use of CS-Enhanced gives to the user becomes it in a tool easy to adopt.

Índice general

Lista de figuras

Lista de tablas

Lista de códigos

Glosario

ACL Access Control List (Lista de control de acceso). Es un control de acceso que consiste en asociar una lista ordenada a cada objeto. Una ACL especifica que usuarios o procesos del sistema tienen otorgado acceso a un determinado objeto y el tipo de operaciones que pueden efectuar sobre él.

DoD Department of Defense (Departamento de Defensa de Estados Unidos) abreviado como DoD o DOD, es el ministerio del gobierno de Estados Unidos encargado de las fuerzas militares del país, todo el tiempo.

Exploit (del inglés to exploit, explotar o aprovechar) es una pieza de software, un fragmento de datos, o una secuencia de comandos con el fin de automatizar el aprovechamiento de un error, fallo o vulnerabilidad, a fin de causar un comportamiento no deseado o imprevisto en los programas informáticos, hardware, o componente electrónico (por lo general computarizado). Con frecuencia, esto incluye cosas tales como la violenta toma de control de un sistema de cómputo o permitir la escalada de privilegios o un ataque de denegación de servicio.

MITRE Es una corporación independiente, sin fines de lucro, comprometida en actividades científicas y técnicas de organizaciones de varios países.

Página web: http://www.mitre.org

NIST National Institute of Stardards and Technology (Instituto Nacional de los Estándares y la Tecnología). Fundado en 1901, es una agencia federal que forma parte del Departamento de Comercio (Department of Commerce) de los EE.UU. La misión del NIST consiste en elaborar y promover patrones de la medición, los estándares y la tecnología con el fin de realzar la productividad, facilitar el comercio y mejorar la calidad de vida. Página web: http://www.nist.gov

NSA National Security Agency (Agencia de Seguridad Nacional). Es una agencia del gobierno de los Estados Unidos responsable de obtener y analizar información transmitida por cualquier medio de comunicación, y de garantizar la seguridad de las comunicaciones del gobierno contra otras agencias similares de otros países. Página web: http://www.nsa.gov/

Sandbox Es un sistema informático de aislamiento de procesos, mediante el cual, se pueden ejecutar distintos programas con seguridad y de manera separada. A menudo se utiliza para ejecutar código nuevo, o software de dudosa confiabilidad, con objeto de evitar la corrupción de datos del sistema en donde estos se ejecutan.

Capítulo 1

Introducción

1.1 Antecedentes

No es posible hablar de seguridad en un sistema computacional sin considerar el importante papel que desempeña el sistema operativo (SO). El SO es el encargado de la administración de los recursos con que cuenta el sistema computacional. Sobre éste están soportadas las funcionalidades para las capas de nivel superior, por lo tanto constituye una parte medular en cuestión de seguridad.

Las organizaciones actuales comparten una gran cantidad de información a través de redes y conexiones a Internet como parte de su actividad diaria, esto es algo inevitable, por lo que se ve la necesidad de regular este intercambio de información para evitar posibles lecturas y modificación no autorizadas. Este tipo de infracciones sobre la información son muy comunes en ambientes compartidos y más si no cuentan con alguna clase de seguridad sobre la información que se maneja. El aseguramiento de la información en una organización tiene que ver con la preservación de los principios de confidencialidad, integridad y disponibilidad sobre dicha información. La manera de lograr mantener estos principios es por medio de la definición e implementación de políticas de seguridad. Una política de seguridad determina los límites de un comportamiento aceptable, y cuales serán las respuestas a las violaciones que se produzcan. Naturalmente, las políticas de seguridad van a ser distintas de una organización

a otra [Cheswick94].

Para abordar esta cuestión, el Departamento de Defensa de Estados Unidos (Department of Defense -DoD) desde hace mucho tiempo ha invertido esfuerzos en el área de sistemas seguros. En particular, en 1985 esa dependencia publicó un documento conocido formalmente como la norma DoD 5200.28-STD, Criterios de Evaluación de la Seguridad de los Sistemas Computacionales (Trusted Computer System Evaluation Criteria -TCSEC), que es conocido comúnmente como "Orange Book" o libro naranja [DoD85]. El libro naranja divide a los sistemas operativos en cuatro categorías (siete subcategorías) con base en sus propiedades de seguridad. El rango de categorías de seguridad de la DoD va de la D (protección mínima) a la A (Protección verificada). El libro naranja ha sido usado para evaluar, clasificar y seleccionar sistemas computacionales considerados para el procesamiento, almacenamiento y recuperación de información sensible o clasificada. Las categorías principales en las que se clasifican los sistemas son:

D – Protección Mínima.

C – Protección Discrecional.

B – Protección Obligatoria.

A – Protección Controlada. Éste es el nivel más alto de seguridad.

Todas las categorías incluyen requisitos tanto de funcionalidad como de confianza, con los cuales se evalúa a los sistemas para comprobar si cumplen con dichos requisitos y saber en que categoría serán puestos [Tanenbaum03]. A continuación se describe con más detalle cada una de las categorías, mencionando algunos de los criterios que deben ser cumplidos en cada una de ellas [DoD85].

Categoría D: Protección mínima. Esta división contiene solo una clase; está reservada para aquellos sistemas que han sido evaluados pero han fallado en cumplir los requerimientos para una evaluación de clase más alta. Ejemplos: MS-DOS, Windows 95 (sin trabajo de grupo) y system 7.x de Apple.

Categoría C: Protección discrecional. En esta división las clases proveen la protección discrecional por medio de capacidades de auditoría, exige la responsabilidad de los

sujetos para las acciones que realizan. Ejemplos de esta clase son: sistemas UNIX, Novell 3.x o superior y Windows NT 4.0.

C1: Protección Discrecional. Limitaciones de acceso a los datos. Protección de nombre de usuario y clave.

C2: Protección de acceso controlado. Archivos de bitácora y de auditoría del sistema.

Categoría B: Protección obligatoria. Esta categoría especifica que los sistemas deben ser obligatorios no discrecionales, esto es que los usuarios o sujetos no pueden cambiar los permisos definidos para los objetos a los que pueden acceder, eso lo definen las políticas de seguridad definidas por el administrador del sistema.

B1: Protección de seguridad por etiquetas. Los sistemas de esta categoría requieren todas las características solicitadas para la categoría C2, además de: seguridad obligatoria y acceso por etiquetas a todos los objetos, verificación de la integridad de las etiquetas, auditoría de objetos etiquetados y control de acceso obligatorio.Soporta seguridad multinivel. Sistemas con este nivel de seguridad se encuentran en agencias de gobierno y del ejército.

B2: Protección estructurada. Esta categoría integra requisitos como: acceso obligatorio sobre todos los objetos y dispositivos, rutas confiables de comunicación entre usuario y sistema, análisis de canales encubiertos, notificación de cambios del nivel de seguridad que afecten a los usuarios y comprobación de la seguridad mejorada.

B3: Dominios de seguridad. Esta categoría debe satisfacer requisitos de herramientas de monitoreo para todos los acceso de los sujetos a los objetos. algunos requisitos son: ruta segura de acceso y autenticación, recuperación segura después de una caída del sistema y auditoría de eventos de seguridad.

Categoría A: Protección verificada. Esta categoría se caracteriza por el uso de métodos de verificación de seguridad formal para verificar que los controles de seguridad discrecional y obligatoria son empleados correctamente. El diseño de seguridad debe estar

revisado y aprobado por un grupo de expertos en seguridad, además todos los componentes del sistema que puedan comprometer la seguridad del mismo, deben provenir de fuentes seguras.

A1: En la práctica, es lo mismo que el nivel B3, pero la seguridad debe estar definida en la fase de análisis del sistema.

Para más información sobre cada una de estas categorías y sus requisitos se puede referir directamente al libro naranja [DoD85].

Actualmente, la responsabilidad sobre la seguridad de sistemas de información la ostenta el Instituto Nacional de los Estándares y la Tecnología (National Institute of Stardards and Technology -NIST [NIST07]) cuya misión consiste en elaborar y promover patrones de la medición, los estándares y la tecnología con el fin de realzar la productividad, facilitar el comercio y mejorar la calidad de vida.

1.2 Planteamiento del problema

La información es un activo muy importante dentro de las organizaciones que debe ser protegido. Parte del problema de que la información quede desprotegida, es el hecho de asumir de manera errónea que la información está segura dentro de las aplicaciones que hacen uso de ella (procesadores de texto, hojas de cálculo, bases de datos, etc), siendo que estas aplicaciones pueden contener fallos en su diseño que exponen a la información a pérdidas o modificaciones indeseadas o indebidas.

El tema de la seguridad empieza a tomar una relevancia cada vez mayor, dado el aumento de la conectividad y compartición de datos a través de las redes de computadoras, por lo que se hace necesario poder contar con sistemas operativos seguros que reduzcan los riesgos a los que se ve expuesta la información en esos ambientes compartidos. Debido a la concientización que se ha hecho al respecto, se han llevado a cabo una gran cantidad de esfuerzos para incrementar la seguridad en los ambientes computacionales; sin embargo, muchos de estos esfuerzos no son muy productivos o adecuados, ya que se basan en la incorrecta suposición de

que la seguridad puede ser provista en el espacio de aplicación sin especificar características de seguridad en el sistema operativo, tal como lo menciona Loscocco en [Loscocco98].

Negar la importancia del sistema operativo para la seguridad del sistema dará como resultado vulnerabilidades en todo el sistema. Todo esto porque es precisamente el sistema operativo el que se encarga de proteger los mecanismos del espacio de aplicación [Loscocco98].

Entre varios controles de acceso que existen (que serán descritos en el capítulo 2) y que pueden ser implementados en el sistema operativo, el control de acceso obligatorio (Mandatory Access Control -MAC), es el único control de acceso que realmente puede proveer un alto nivel de seguridad, esto es porque MAC controla el acceso a los recursos por medio de políticas, las cuales son definidas por el administrador del sistema no por el dueño del recurso, estas políticas definen de manera explícita los accesos autorizados a los recursos y las operaciones permitidas sobre éstos. El principal inconveniente con MAC, es la complejidad de su implementación y configuración, lo que hace que sea una alternativa poco utilizada.

De acuerdo a la Agencia de Seguridad Nacional (National Security Agency -NSA [NSA07]) de los Estados Unidos, Security-Enhanced Linux, SELinux o Linux con seguridad mejorada es un conjunto de parches para el kernel de Linux y algunas utilerías para la incorporación de una arquitectura de control de acceso obligatorio robusto y flexible dentro de la mayoría de los subsistemas del kernel, esto quiere decir que SELinux incluye MAC. SELinux provee un mecanismo de seguridad mejorado para reforzar la separación de la información basado en los requisitos de confidencialidad e integridad, el cual permite que los intentos de eludir o forzar a los mecanismos de seguridad de la aplicación sean detectados y se confine el daño, posiblemente causado por aplicaciones defectuosas.

Linux cuenta con una herramienta de firewall (cortafuegos) llamada IPtables [Netfilter09], y es usada para el filtrado de paquetes de la red entre otras funciones. IPtables viene incluida en prácticamente todas las distribuciones de Linux actuales [Purdy04]. Esta herramienta trabaja en el espacio de usuario y el filtrado de red se define por medio de políticas.

Con base en lo expuesto anteriormente sobre MAC, y la razón por la cual no se le utiliza, la propuesta que se hace para resolver este problema, es la de crear una interfaz de software (CS-Enhanced) que permita llevar a cabo la implementación de MAC de SELinux

(específicamente sobre aplicaciones que trabajan bajo la arquitectura Cliente-Servidor) de una forma que resulte sencilla, fácil y efectiva. Pero además esta interfaz va a conjuntar MAC con las reglas de filtrado de IPtables para controlar el acceso sobre los objetos de la red.

La implementación de MAC de SELinux para una aplicación es por medio de la creación de una política, que se instala en el kernel, cada política será distinta de acuerdo a la aplicación en cuestión. Para este caso particular de una aplicación Cliente-Servidor, se crea una política para el cliente y una política para el servidor. Lo que ambas políticas deben hacer es controlar que solo quien esté explícitamente autorizado para ejecutar al cliente, al servidor o ambos pueda hacerlo. Las reglas de filtrado del firewall de IPtables deben especificar que tráfico de red está permitido entre el cliente y el servidor, además de definir el puerto de red que se utilizará y que va a quedar reservado para este cliente y este servidor.

De esta forma se aplica el control de acceso obligatorio y el filtrado de red. El cliente, el servidor y el puerto quedan confinados, en el sentido de que se les proporciona un entorno de seguridad, en donde pueden operar sin interferir con algún otro proceso y que en caso de que presente algún problema con la aplicación, éste no se extenderá al resto del sistema, además este confinamiento evita que la aplicación Cliente-Servidor pueda ser usada como puerta de entrada para Troyanos buscan ganar privilegios (como convertirse en root) para acceder a otros recursos del sistema y sacar provecho de ello.

Básicamente la interfaz hará que SELinux e IPtables trabajen de manera conjunta y coordinada, vinculandolos a través de la política creada para que por medio de la implementación del control de acceso obligatorio y el filtrado de red se mejore de manera efectiva la seguridad de la aplicación Cliente-Servidor que se ejecuta. Ver Figura 1.1.

1.3 Objetivos de la tesis

1.3.1 Objetivo general

El objetivo general de la tesis es el desarrollo de una herramienta que facilite la implementación de un control de acceso obligatorio en una arquitectura Cliente-Servidor.

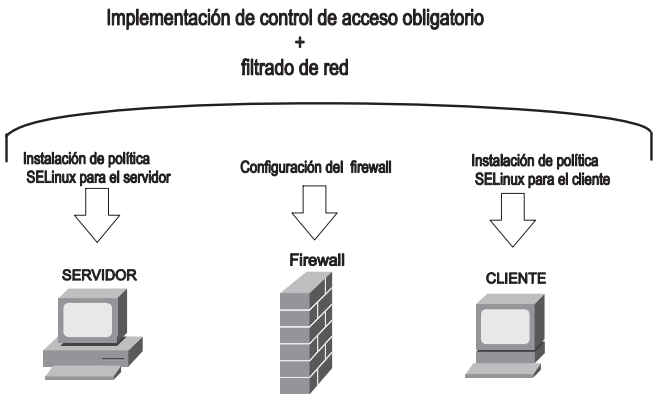

Figura 1.1: Implementación del control de acceso obligatorio con CS-Enhanced

1.3.2 Objetivo particular

Partiendo de la definición de una política de seguridad para un cliente y un servidor hecha por el usuario, traducir esta definición a una implementación de control de acceso obligatorio por medio de la generación de diversos archivos de bajo nivel en SELinux e IPtables. CS-Enhanced deberá generar todos los archivos necesarios, así como configurar distintos elementos tanto de SELinux como de IPtables para implementar de manera correcta y completa el control de acceso obligatorio sobre la arquitectura Cliente-Servidor. El manejo de CS-Enhanced debe proporcionar sencillez al usuario administrador al grado que no sea necesario que éste requiera de conocimientos previos en SElinux o IPtables (Ver Figura **??**).

1.3.3 Justificación

A pesar de que SELinux implementa un control de acceso obligatorio en el kernel de Linux, el cual le proporciona seguridad adicional sobre los objetos del sistema [Smalley05], [Smalley01], hacer uso de este control de acceso implica un importante trabajo de desarrollo, prueba e implementación de políticas en el lenguaje de SELinux [Coker08].

Además debe mencionarse que para los objetos que interactúan con otros objetos a través de una red, intercambiando información (como son los que trabajan bajo la arquitectura Cliente-Servidor), una política SELinux no les puede proporcionar protección en ese

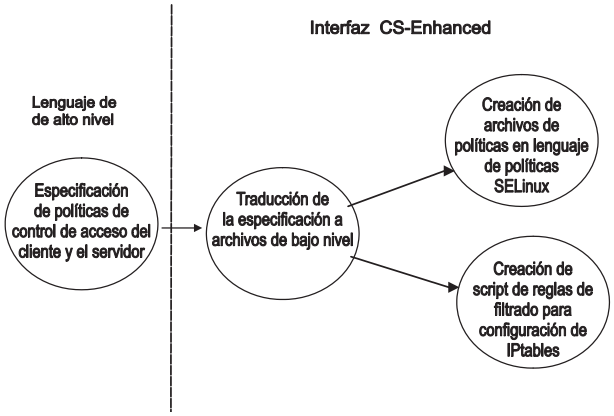

Figura 1.2: descripción de la herramienta desarrollada

nivel, a pesar de ser necesario. Aunque IPtables hace el filtrado efectivo de paquetes en la red de acuerdo a las reglas que le sean definidas en su configuración, éste firewall no sabe de aplicación de políticas, ni etiquetado de objetos, aplicando sus reglas de filtrado en los paquetes manejados, sin distinguir entre sujetos, objetos y permisos de acceso.

Es por eso que, analizando lo anterior, desarrollamos una herramienta de software (interfaz CS-E) para que por un lado se logre un refuerzo de la seguridad de la aplicación Cliente-Servidor, por medio del trabajo conjunto entre SELinux e IPtables, y por el otro promover la utilización de MAC, facilitando su uso, dado que consideramos que es el único control de acceso que realmente puede ofrecer un alto grado de seguridad a las aplicaciones sobre las que se implementa.

1.4 Descripción de los capítulos

En el Capítulo 1 se hace una reseña de los antecedentes sobre la importancia de seguridad de la información y los esfuerzos hechos para tratar de mantenerla. Se enlista, haciendo una breve descripción de las diferentes categorías en las cuales se clasifica a los sistemas computacionales de acuerdo a una serie de criterios establecidos, en una norma emitida por el Departamento de Defensa de los Estados Unidos.

En el Capítulo 2 se hace una descripción de lo que es un control de acceso, los diferentes tipos de controles de acceso; se hace una descripción de cada uno de ellos, sus características, funcionamiento y aplicación.

El Capítulo 3 describe lo que es SELinux, y su operación. Se mencionan los controles de acceso obligatorio con que cuenta, la manera en como se aplican las decisiones de acceso definidas en las politicas SELinux, parte de la sintaxis del lenguaje de politicas, clase de objetos que lo componen y algunos comandos de Linux modificados para su utilización en SElinux.

En el Capítulo 4 se describe lo que es la arquitectura Cliente-Servidor e IPtables. Se describe su funcionamiento, sus reglas de uso y ejemplos.

En el Capítulo 5 se explica todo lo relacionado con la herramienta de software desarrollada: las partes que la conforman, como funciona cada una de ellas y el resultado final del trabajo en conjunto de todas las partes. Se presenta un ejemplo práctico y real, para demostrar su aplicación.

En el Capítulo 6 se presentan las conclusiones y los trabajos futuros.

Capítulo 2

Control de acceso

Los controles de acceso son un medio para restringir el acceso a los recursos de un sistema y también una manera de intentar salvaguardar estos recursos de amenazas potenciales [Sandhu94]. Los controles de acceso han surgido a lo largo del tiempo, evolucionando y adaptandose a las necesidades de protección específicas de distintos ámbitos (militar, comercial), buscando proteger la infomación de una forma mas adecuada y eficaz. Los controles de acceso como se verá en este capítulo tienen formas distintas de operación, pero comparten la misma finalidad, que es la de resguardar la información de las amenazas potenciales.

2.1 Control de acceso

Un control de acceso o autorización es un mecanismo de seguridad utilizado para proteger o limitar el acceso a propiedades u objetos de valor. Trasladando este concepto al área computacional, el control de acceso forma parte del esquema básico de seguridad que todo sistema debe tener, en este caso para proteger y administrar el acceso a los recursos compartidos del sistema, el control de acceso juega un papel crítico al proveer seguridad a los niveles más altos.

Los controles de acceso determinan cuándo y cómo puede ser usado un recurso y que operaciones se tiene permitido hacer sobre él; como por ejemplo: iniciar sesión en un servidor solo

en determinadas horas, acceder a ciertos directorios o ejecutar determinados programas, etc. los anteriores son ejemplos concretos de operaciones aplicadas a recursos.

El control de acceso es solo un aspecto que forma parte de lo que constituye una solución computacional de seguridad, pero es uno de los más notorios. Cada vez que un usuario inicia sesión en un sistema computacional multiusuario, el control de acceso es ejecutado [Ferraiolo03].

2.1.1 El propósito y fundamentos del control de acceso

Para poder entender la importancia que juegan los controles de acceso en la seguridad de la información, es necesario mencionar cuales son los principios básicos de la seguridad de la información, de acuerdo con Ferraiolo [Ferraiolo03] son los siguientes:

Confidencialidad. Se refiere a la necesidad de mantener la información segura y privada. Esta categoría puede incluir cualquier cosa, desde secretos de estado hasta memorándums confidenciales, información financiera e información de seguridad como las contraseñas. El control de acceso es crítico para conservar la confidencialidad y la integridad de la información. La condición de confidencialidad requiere que solo usuarios autorizados puedan leer información.

Integridad. Se refiere al concepto de proteger la información de ser alterada o modificada inadecuadamente por usuarios no autorizados. La condición de integridad requiere que solo los usuarios autorizados puedan alterar la información en formas permitidas.

Disponibilidad. Se refiere a la noción de que la información debe estar disponible para usarse cuando se necesite.

En la Tabla 2.1 se muestran los principios básicos de seguridad y las amenazas a las que se ven expuestos estos principios.

2.1.2 Autenticación y autorización

Los sistemas hacen uso de diferentes medios para determinar si un usuario es quien en verdad dice ser, esto es: autenticarse en el sistema. De acuerdo a Ferraiolo [Ferraiolo03], esta autenticación se basa en uno o más de los siguientes factores:

- Algo que se sabe, como la contraseña, el número de identificación personal (NIP), una combinación. etc.
- Algo que se posee, como una tarjeta inteligente, una tarjeta de cajero automático o una llave.
- Algo que se es, o alguna característica física que se posee, como la huella digital, un patrón retinal o una característica facial.

La autenticación es normalmente más fuerte si se usan dos o más factores. Una vez aprobada la autenticación, es decir, se probó la identidad del usuario, la autorización específica que es lo que el usuario en cuestión tiene permitido hacer dentro del sistema.

2.1.3 Sujetos, objetos, operaciones y permisos

A continuación se definen algunos términos clave utilizados en los controles de acceso.

- **Sujeto**. Un sujeto es un proceso o tarea que opera en nombre del usuario dentro del sistema computacional.
- **Objeto**. Recurso disponible en el sistema computacional. Entidad del sistema sobre la cual puede ser llevada a cabo una operación. Un objeto puede ser: un archivo, un directorio, un puerto TCP/UDP, segmentos de memoria compartida, etc.

Principio	**Amenaza**
Confidencialidad de los datos	Revelación de los datos
Integridad de los datos	Alteración de los datos
Disponibilidad del sistema	Negación del sistema

Tabla 2.1: Principios de la información y sus amenazas

- **Operación.** es un proceso activo invocado por un sujeto.

- **Permisos**. Son las autorizaciones que permiten llevar a cabo alguna acción en el sistema. Un permiso se refiere a la combinación de un objeto y una operación. Una operación particular usada en dos diferentes objetos representa dos permisos distintos, y de forma similar, dos operaciones distintas, aplicadas a un objeto simple representan dos permisos distintos [Ferraiolo03].

Los sujetos son entidades activas que actúan sobre los objetos iniciando peticiones para llevar a cabo operaciones sobre los objetos. Los objetos son vistos comúnmente como entidades pasivas que contienen o reciben información, (ver Figura 2.1).

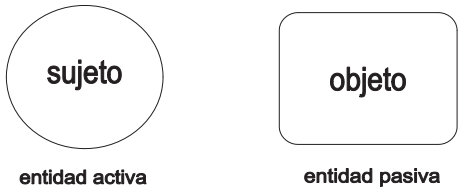

Figura 2.1: Sujeto y objeto

2.1.4 Principio de privilegio mínimo

Se refiere a que a las aplicaciones se les otorgan solo los permisos necesarios para poder realizar su tarea, de tal manera que en caso de que se produzca algún problema con la aplicación, ésta quedará confinada y el daño no afectará a otras partes del sistema. Bishop [Bishop02] lo define como: El principio que restringe la manera en que los privilegios son concedidos. Establece que a un sujeto le deben ser concedidos solo aquellos privilegios que necesita para llevar a cabo sus tareas y que si un sujeto no necesita un permiso de acceso, no se le debe conceder.

2.1.5 Interacción sujeto-objeto

Los sujetos y los objetos cuentan con un conjunto de atributos de seguridad. Para permitir que un sujeto acceda a un objeto determinado, el mecanismo de control de acceso debe hacer una comparación entre los atributos de seguridad del sujeto y los atributos de seguridad del objeto que intenta ser accedido, esto de acuerdo a un conjunto de reglas predeterminado, de acuerdo al resultado de ésta comparación, se permite o no el acceso al objeto.

2.2 Modelos de control de acceso

Entre los principales modelos de control de acceso están los que se mecionan a continuación:

- Modelo de control de acceso discrecional.
- Modelo de control de acceso obligatorio.
- Modelo de control de acceso de matriz de acceso.
- Modelo de control de acceso Bell-Lapadula.
- Modelo de control de acceso Biba.
- Modelo de control de acceso cumplimiento de tipo (TE).
- Modelo de control de acceso basado en roles (RBAC).

2.2.1 Modelo de control de acceso discrecional (DAC)

La definición del control de acceso discrecional de acuerdo al DoD en [DoD85] es la siguiente: un medio de restringir el acceso a los objetos basándose en la identidad de los sujetos y/o grupos a los cuales pertenecen. Los controles son discrecionales en el sentido de que un sujeto con ciertos permisos de acceso es capaz de transferir esos permisos (quizás indirectamente) a cualesquier otros sujetos. Podría decirse que el sujeto o entidad que transfiere los permisos es el dueño del objeto, por lo que en este modelo, el sujeto tiene el control completo sobre los objetos que le pertenecen.

De acuerdo a Ferraiolo en [Ferraiolo03], DAC permite que el otorgamiento o revocación de permisos de acceso sea puesto a discreción de los usuarios individuales. Un mecanismo DAC permite a los usuarios el revocar o el otorgar acceso a cualquiera de los objetos bajo su control sin la intervención de un administrador del sistema.

En DAC la seguridad del sistema depende de las aplicaciones que estén siendo ejecutadas, y por lo tanto cuando un hueco de seguridad en una aplicación tiene lugar, esto puede afectar la seguridad de todos los objetos a los cuales el usuario tiene acceso, esto hace que DAC sea muy vulnerables a los caballos de Troya [Lehtinen91]

DAC es un modelo de control de acceso ampliamente utilizado. Ejemplo del uso de este modelo se aprecia en los sistemas UNIX, en la utilización de *bits de protección* para definir los permisos que se tienen sobre determinado recurso del sistema, por parte del dueño, el grupo o el resto de los usuarios que tienen acceso a él. Un archivo llamado *file1* que tiene sus bits de protección de la siguiente manera:

```
file1   rwx   r-x   --x
```

Nos esta indicando con esa configuración que: el dueño del archivo puede leerlo (r), escribirlo o modificarlo (w) y ejecutarlo (x); el grupo puede leerlo, no puede modificarlo (el guion (-) indica que no se posee el permiso) y puede ejecutarlo; el resto de lo usuarios, es decir los que no son ni el dueño ni pertenecen al grupo, solo pueden ejecutarlo.

2.2.2 Modelo de control de acceso obligatorio (MAC)

El control de acceso obligatorio (mandatory access control (MAC)) de acuerdo al TCSEC es un medio de restringir el acceso a los objetos basándose en la sensibilidad (representada como una etiqueta) de la información contenida y la autorización formal (esto es, autorización) de los sujetos para acceder información de tal sensibilidad. El modelo de control de acceso obligatorio se ubica en la categoría B1 del libro Naranja [DoD85].

Un responsable (administrador) clasifica los objetos y sujetos según sus respectivos niveles de seguridad y los divide según el principio de mínimo privilegio.

Solo los mecanismos de seguridad obligatoria pueden proveer rigurosamente garantías

de integridad. Estos mecanismos deben ser usados para garantizar que los mecanismos de seguridad son aplicados como se requiere y pueden proteger al usuario en contra de ejecuciones inadvertidas de aplicaciones no confiables [Loscocco98].

En MAC se definen políticas, las cuales solo pueden ser modificadas por el administrador. Estas políticas incrementan el nivel de seguridad del sistema, ya que no permite ninguna operación que no este explícitamente autorizada por un administrador.

Este modelo está desarrollado para ser implementado en sistemas en el cual la confidencialidad tiene la más alta prioridad, como en el ejército. Los sujetos reciben una etiqueta de autorización y los objetos reciben una etiqueta de clasificación, referida también como niveles de seguridad.

El término "obligatorio" usado con los controles de acceso históricamente implica un muy alto grado de robustez que asegura que el mecanismo de control resiste subversión.

Siempre que un sujeto intente acceder a un objeto, una regla de autorización aplicada por el kernel del sistema operativo examina los atributos de seguridad y decide si el acceso puede tener lugar. Cualquier operación hecha por cualquier sujeto sobre cualquier objeto será probada contra el conjunto de reglas de autorización para determinar si la operación es permitida.

Los elementos básicos que componen los modelos de seguridad MAC son: sujetos, objetos, modos de acceso y niveles de seguridad, en donde los modos de acceso son los permisos que tienen los sujetos para operar sobre los objetos y los niveles de seguridad son los permisos de los sujetos y la clasificación de los objetos.

La relación de los modos de acceso con los sujetos y objetos es determinada por una estructura matricial basada en el modelo de matriz de acceso que se desarrolló en los años 70. La estructura matricial se utiliza en los modelos de seguridad para definir el conjunto de operaciones que un sujeto puede hacer sobre un objeto.

Como una racionalización en el TCSEC, MAC soporta requerimientos del DoD y regulaciones pertinentes de accesos no autorizados a información clasificada, y en particular a la protección de la confidencialidad (lectura u observación) de información sensible.

Los sistemas que soportan las políticas MAC están preocupados por el flujo ilícito de

un alto nivel a un bajo nivel, como tal, el soporte de la política es con respecto a controlar la lectura y la escritura. Sin embargo el control sobre las operaciones de escritura solo se encarga de prevenir la observación ilícita indirecta de información sensible y no de su integridad (modificación o destrucción no autorizada)

2.2.3 Modelo de control de acceso de matriz de acceso

Este modelo está basado en los conceptos de: sujetos, objetos, permisos y dominios. En la Tabla 2.2 se representa una matriz de acceso cuyas filas representan dominios (o sujetos) y las columnas representan objetos. Las entradas de la matriz consisten en una serie de derechos de acceso.

Un dominio se refiere a un par objeto-derechos en la matriz de acceso, cada par especifica un objeto y algún subconjunto de las operaciones que pueden ejecutarse en él (leer, escribir, ejecutar).

	Archivo1	**Archivo2**	**Impresora1**	**Tabla1**
Dominio1	Leer/Escribir			
Dominio2		Leer		Leer/Escribir
Dominio3		Ejecutar	Escribir	

Tabla 2.2: Matriz de acceso

Mientras que la matriz de acceso es raramente implementada, dado que es grande y tiene pocos elementos ocupando una gran cantidad de espacio en memoria, este modelo es útil para entender el comportamiento y las propiedades de los demás modelos de control de acceso [Saunders01] .

2.2.4 Modelo de control de acceso Bell-Lapadula

Este modelo fue desarrollado en el año de 1976 por David Elliott Bell y Len LaPadula [Bell73], en la organización MITRE, la cual fue creada por el gobierno de Estados Unidos para

elaborar estudios sobre modelos de seguridad militar. Bell y LaPadula formalizaron las reglas de un control de acceso militar en un modelo matemático adecuado para definir y evaluar sistemas de seguridad computacional. Este modelo es un control de acceso obligatorio.

Los sujetos y los objetos son etiquetados con niveles de seguridad (NS), de acuerdo a los cuales se determinan los derechos de acceso. Los niveles manejados en el entorno militar son: "no clasificado", "confidencial", "secreto" y "secreto máximo", siendo este último el nivel más alto de la jerarquía.

A los sujetos se les maneja el nivel de seguridad como nivel de permiso, dependiendo de que documentos estén autorizados para ver, a los objetos se les maneja como nivel de sensibilidad, esto de acuerdo a la importancia que tiene la información contenida por este.

Se hace uso de una matriz de acceso discrecional basada en el modelo de matriz de acceso para definir los permisos o modos de acceso que tienen los sujetos sobre los objetos como se muestra en la Tabla 2.3, donde los permisos manejados en la matriz de acceso son:
Leer: El sujeto puede leer el objeto pero no modificarlo.
Agregar : El sujeto puede escribir el objeto pero no puede leerlo.
Ejecutar : El sujeto puede ejecutar el objeto pero no puede leerlo directamente.
Leer/Escribir: El sujeto puede leer y escribir el objeto.

		Objetos	
		Archivo X	Archivo Y
Sujetos	X	Leer	Leer/Escribir
	Y	-	Ejecutar

Tabla 2.3: Matriz de acceso modelo Bell Lapadula

Las siguientes propiedades rigen el flujo de información en el modelo:
Propiedad 1. Propiedad de seguridad simple: ningún sujeto puede tener acceso de lectura a ningún objeto que tenga un nivel de seguridad mayor que la autorización del sujeto. Por ejemplo, un coronel puede leer los documentos de un capitán, pero no viceversa, ya que

el coronel tiene un rango mayor al del capitán, que de acuerdo al modelo se traduciría en un nivel de seguridad mayor para el coronel.

Propiedad 2. Propiedad estrella (*) ningún sujeto puede tener permiso de agregar (escribir) en un objeto cuyo nivel de seguridad no es al menos igual al nivel actual de seguridad del sujeto. Ningún sujeto puede tener acceso de lectura-escritura a un objeto cuyo nivel de seguridad no es igual al actual nivel de seguridad del objeto. Por ejemplo, un capitán puede anexar un mensaje al buzón de un teniente diciendo todo lo que sabe con respecto a una estrategia militar, pero un coronel no puede anexar un mensaje al buzón del capitán diciendo todo lo que sabe al respecto de dicha estrategia, porque la información que el coronel tiene puede ser información de secreto máximo que de ninguna manera puede ser revelada a un capitán. La Figura 2.2 (tomada de [Tresys07]) muestra como se aplican estas propiedades para controlar el flujo de información entre los niveles.

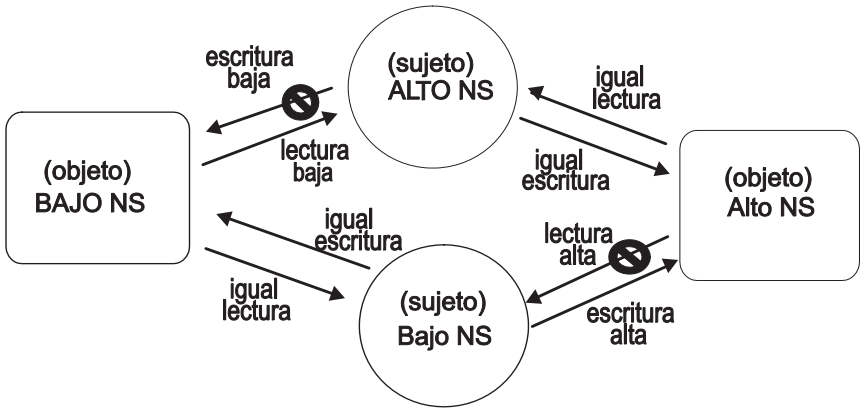

Figura 2.2: Propiedades modelo Bell-Lapadula.

Este modelo busca preservar la confidencialidad de la información y no tiene en cuenta otros principios como integridad o disponibilidad. Puesto que hay múltiples niveles de seguridad este modelo también se le denomina seguridad multinivel.

Un ejemplo de esto sería: personal no clasificado no puede leer datos de niveles confidenciales, y datos de un nivel de secreto máximo no pueden ser escritos en archivos de niveles no clasificados.

2.2.5 Modelo de control de acceso Biba

Este modelo fue creado por K. J. Biba en el año de 1977 para el MITRE. Este modelo esta orientado a la integridad de los datos. De igual forma que el modelo de Bell-Lapadula, Biba clasifica a los objetos y usuarios por niveles de seguridad.

Este modelo se centra en proteger la integridad de los datos sobre la protección contra usuarios de un nivel inferior que pueden escribir en cualquiera de los niveles superiores en los cuales está ubicado. Lo cual permite a un usuario de un bajo nivel de seguridad sobreescribir documentos altamente secretos, a menos que se adoptara un conjunto de políticas de integridad.

Biba es un modelo de integridad que supone una separación en *niveles de integridad* (NI) con una relación ordenada. Los objetos son asignados a clases de integridad de acuerdo al daño que sufrirían si fueran modificados de manera inapropiada, y los usuarios asignados a clases de integridad basados en su veracidad.

Para garantizar la integridad de los datos se deben observar las siguientes propiedades:

Propiedad 1. Propiedad de integridad simple. Un objeto que se ejecuta en un determinado nivel de seguridad, solo puede escribir objetos de su nivel o de un nivel inferior. Tambien se le conoce como propiedad de no escritura hacia arriba.

Propiedad 2. **Propiedad de integridad *** . Un sujeto que se ejecuta en un determinado nivel de seguridad solo puede leer objetos de su nivel o un nivel superior, no puede escribir en ellos. También se le conoce como propiedad de no lectura hacia abajo.

Resumiendo ambas propiedades, indican que un sujeto no puede leer a un objeto de menor integridad que el suyo y tampoco puede escribir a un objeto con un nivel de integridad mayor al suyo. La Figura 2.3 (tomada de [Tresys07])muestra como se aplican estas propiedades para controlar el flujo de información entre los niveles.

Si quisieramos aplicar este modelo a una compañia, definiriamos algunos niveles, por ejemplo, el jefe de compañia, el gerente del área de manufactura y el mensajero, teniendo el jefe el nivel mayor y el menor el mensajero. Aqui por ejemplo nos interesa lo que el jefe tenga que comunicarle a través de un memorandum al gerente del área sobre una decisión

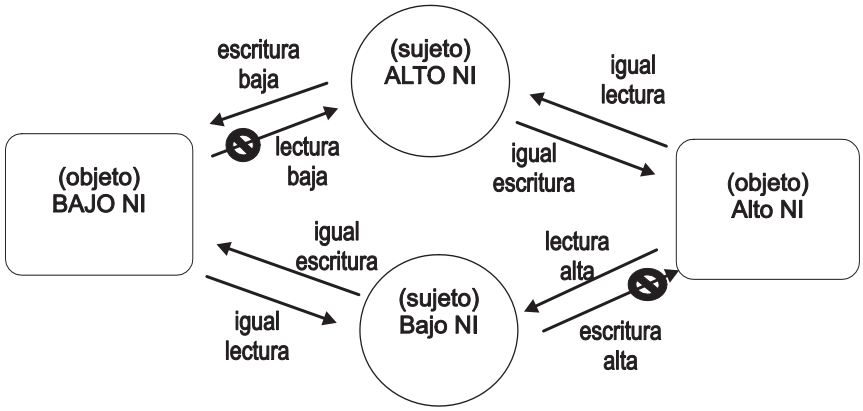

Figura 2.3: Propiedades modelo Biba.

tomada y que éste debe ejecutar. Otro ejemplo sería el de un comunicado por parte de la dirección de una escuela acerca de cual es el reglamento a seguir el siguiente ciclo escolar, este documento va dirigido a los alumnos, quienes lo podrán leer pero no lo podrán modificar, en este caso el documento (objeto) tiene un nivel de integridad mayor que los alumnos (sujetos), por lo que se aplica la propiedad 2.

2.2.6 Modelo de control de acceso de cumplimiento de tipo (TE)

De acuerdo a la definición de la NSA [NSA07], el modelo de cumplimiento de tipo o TE asocia a cada sujeto un atributo de seguridad llamado dominio y a cada objeto uno llamado tipo. TE es un mecanismo de control de acceso obligatorio orientado a tablas, es apropiado para confinar aplicaciones y restringir flujos de información [Badger95].

El modelo tradicional de TE agrupa a todos los sujetos en un mismo dominio y a todos los objetos en un mismo tipo. Un par de matrices de acceso especifican como los dominios acceden a los tipos y como los dominios pueden interactuar con otros dominios. Cada usuario está autorizado a operar en ciertos dominios.

Modelo cumplimiento de dominio-tipo (DTE)

Ferraioloen [Ferraiolo03] describe al modelo de cumplimiento de dominio-tipo (DTE) de la siguiente manera: El modelo DTE es una abstracción de los conceptos involucrados en el mecanismo DTE. El mecanismo DTE es una mejora del mecanismo de cumplimiento de tipo, el cual es un mecanismo de control de acceso orientado a tablas, desarrollado por Boebert y Kain en los 80´s. DTE provee los beneficios de TE pero reduce los costos y la complejidad [Badger96].

El modelo DTE como muchos otros modelos de control de acceso, divide el sistema computarizado en dos entidades lógicas: sujetos y objetos, los sujetos son entidades activas (usualmente procesos). Los objetos son entidades pasivas (por ejemplo: archivos, directorios, dispositivos, y segmentos de memoria). Un dominio está asociado con un sujeto. Un tipo está asociado con un objeto .

La asignación de una etiqueta de "dominio" para un sujeto generalmente depende de su función (por ejemplo, una transacción de un proceso de negocios). Un objeto es asignado a un tipo basado en sus requerimientos de integridad. Los permisos de control de acceso son asociados con ambos dominios y tipos. Esto ocasiona 2 tipos de permisos: permisos dominio-dominio y permisos dominio-tipo. Cada uno de estos tipos de permisos está representado usando una tabla de correspondencia de tipos. La tabla de control de acceso dominio-dominio (DDAT, por sus siglas en inglés) es una tabla de dos dimensiones con una entrada para cada par ordenado de dominios. Similarmente la tabla de control de acceso de dominio-tipo (DTAT, por sus siglas en ingles) es una tabla de dos dimensiones para cada par dominio-tipo. Ya que puede haber más de un permiso asociado con un par dominio-dominio o un par dominio-tipo, cada entrada en estas tablas contiene un conjunto de permisos. Juntas todas las entradas en estos dos tipos de tablas constituyen la base de datos DTE para un ambiente.

Ejemplos de permisos dominio-dominio (en el contexto de un sistema operativo UNIX) son create(C) y kill (K). Ejemplos de permisos dominio-tipo son read(R), write (W), execute (E), y browse directory (T).

Los permisos dominio-dominio son creados para expresar interacciones permitidas entre

sujetos. Por ejemplo, un sujeto A (proceso A) puede crear una instancia de otro proceso (proceso B) solo si existe una entrada crear entre el dominio del sujeto A y el domino del sujeto B.

2.2.7 Modelo de control de acceso basado en roles (RBAC)

Este modelo es propuesto en 1992 por David Ferraiolo and Rick Kuhn para afrontar el reto que representa la administración de la seguridad en redes de gran tamaño. Sus principios se basan en el concepto de rol usado por las organizaciones en el mundo real, el cual define las funciones específicas que una persona en un puesto determinado debe cumplir dentro de la organización. RBAC toma ese concepto y lo aplica para definir permisos (funciones) permitidos sobre los objetos del sistema. Los roles cuentan con diferentes privilegios y responsabilidades que han sido bien identificados en las organizaciones de negocios, y las aplicaciones computacionales.

Dentro de una organización los roles son relativamente estables, mientras que los usuarios y permisos son ambos numerosos y pueden cambiar rápidamente, este modelo presenta la ventaja de que al poder agrupar a los usuarios en roles no es necesario definir de manera individual los permisos para cada nuevo usuario del sistema computacional, sino que al momento de asignar un rol a un sujeto, éste automáticamente puede hacer uso de los permisos asignados a dicho rol; esto libera al administrador del sistema de una significativa carga de trabajo que tendría si tratara a cada usuario de forma individual. La Tabla 2.4 muestra la descripción formal original para RBAC , en ella se enuncia que:

1. El rol activo de un sujeto, es aquel que está usando actualmente.
2. Cada sujeto puede estar autorizado para ejecutar uno o más roles y
3. Que cada rol puede estar autorizado para ejecutar una o más transacciones.

Dado lo anterior se presentan 3 reglas básicas requeridas, las cuales son:

Regla 1. Asignación de rol. Un sujeto puede ejecutar una transacción solo si el sujeto ha sido asignado a un rol. El proceso de identificación y autenticación no es considerado una transacción, todas las demás actividades del usuario en el sistema son conducidas a través de

transacciones. Por lo tanto, todos los usuarios activos requieren tener algún rol activo.

Regla 2. Autorización de rol. El rol activo de un sujeto debe ser autorizado para el sujeto. Con la regla 1, esta regla asegura que los usuarios pueden asumir solo roles para los cuales están autorizados.

Regla 3. Autorización de transacción. Un sujeto puede ejecutar una transacción solo si la transacción está autorizada para el rol activo del sujeto, en conformidad con las reglas 1 y 2, esta regla asegura que los usuarios pueden ejecutar solo transacciones para las cuales están autorizados. Una caracteristica clave de este modelo es que todos los accesos son a través de roles, un rol es esencialmente una colección de permisos, y todos los usarios reciben permisos exclusivamente a través de los roles a los cuales son asignados tal como se muestra en la Figura 2.4

Descripción formal de RBAC
Para cada sujeto, el rol activo es el único que el sujeto está usando actualmente: AR(*s: sujeto*)= { el rol activo para el sujeto s}
Cada sujeto puede estar autorizado para ejecutar uno o más roles: RA(*s: sujeto*)= {roles autorizados para el sujeto s}
Cada rol puede ser autorizado para ejecutar una o más transacciones: TA(*r: rol*)= {transacciones autorizadas para el rol r}
Los sujetos pueden ejecutar transacciones. El predicado *exec(s,t)* es verdadero si y solo si el sujeto s puede ejecutar la transacción t en el momento actual; en otro caso es falso: *exec(s: sujeto, t: tran*)= {verdadero si y solo si el sujeto s puede ejecutar la transacción t} **1.Asignación de rol**: un sujeto puede ejecutar una transacción solo si el sujeto tiene seleccionado o le ha sido asignado un rol: $\forall$*s: sujeto, t: tran* · *exec(s,t)* AR $(s) \neq \varnothing$ **2.Autorización de rol**: un rol activo de un sujeto debe estar autorizado por él sujeto: $\forall$*s: sujeto*·*AR (s)* $\subseteq$ RA(s) **3.Autorización de transacción**: un sujeto puede ejecutar una transacción solo si la transacción es autorizada por el rol activo del sujeto: $\forall s$ *: sujeto, t: tran* · *exec(s,t)* $\Longrightarrow$ $t \in$TA (AR (s))

Tabla 2.4: Descripción formal RBAC

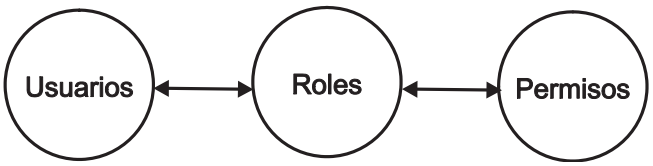

Figura 2.4: Relaciones en RBAC

El método más común para implementar un control de acceso en un sistema computacional es por medio de una lista de control de acceso (ACLs). Todos los recursos del sistema, tales como archivos, impresoras, y terminales tienen adjuntada una lista de usuarios autorizados. Podriamos preguntarnos algo como ¿Qué usuarios tienen acceso al objeto x ? contestar esta pregunta es fácil, mucho más difícil es contestar la pregunta: ¿A qué objetos puede tener acceso el usuario x?, para responderla se requiere revisar todos los objetos en el sistema computacional, el cual puede ser de millones, grabar sus listas de control de acceso y finalmente reportarlo sobre el usuario x.

Un efecto secundario de este esquema es que las ACLs facilitan el agregar permisos a un objeto pero es difícil revocar todos los permisos de un usuario en particular. En muchos sistemas, los usuarios están combinados dentro de grupos, los cuales son usados entonces como entradas en las ACLs.

RBAC y el mecanismo convencional de grupos (como el que se maneja en los sistemas UNIX, Linux) son de algún modo similares. Un grupo es una colección de usuarios, más que una colección de permisos, y los permisos pueden ser asociados a ambos, usuarios y los grupos a los cuales ellos pertenecen, como se muestra en la Figura 2.5 en donde los permisos son asociados a los usuarios a través de los grupos o bien directamente, y una vez asociados pueden acceder a los objetos basándose ya sea en su identificador de usuario o su identificador de grupo; por lo que es posible que un usuario retenga permisos de acceso que debieron ser revocados cuando el grupo al que este pertenece dejo de tener permisos sobre el objeto.

Los permisos basados en el identificador del usuario son en efecto un hueco en el cum-

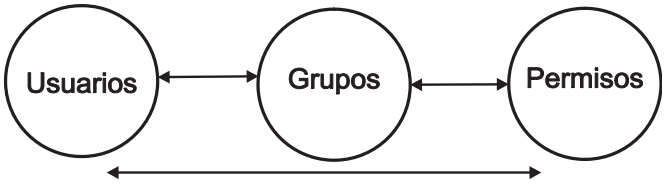

Figura 2.5: Relaciones del control de acceso a grupos

plimiento de la política de seguridad. El requisito de RBAC de que todos los accesos sean a través de roles ayuda a reforzar la seguridad significativamente en las aplicaciones reales eliminado este hueco de seguridad [Ferraiolo03].

Mientras que los grupos normalmente son tratados como colecciones horizontales de usuarios, RBAC ofrece otra característica importante que es el hecho de que los roles sean jerárquicos, esto es, los roles pueden heredar permisos de otros roles, tal como se muestra en la Figura 2.6, en donde se esta especificando una jerarquía por roles en el área médica, en donde el residente tiene una jerarquia menor que el médico y este a su vez se ubica por debajo de los especialistas (cardiólogo, oncólogo etc), tal como sería en la vida real.

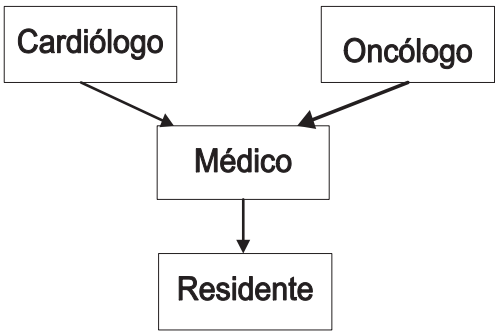

Figura 2.6: Ejemplo de una jerarquía funcional de roles en RBAC

2.3 Conclusiones

Controles de acceso como DAC y RBAC siguen estando vigentes y se siguen implementando en los sistemas operativos, ya sea de manera independiente o en conjunto como una forma de reforzar la seguridad de estos. Algunos controles de acceso (como MAC) no son muy utilizados a pesar del alto grado de seguridad que ofrecen, debido a la complejidad de su implementación y configuración, por lo que muchas veces se retorna a controles de acceso más sencillos de implementar pero menos seguros, que exponen a la información a amenazas de seguridad que muy probablemente no será capaz de salvaguardar.

Capítulo 3

SELinux

SELinux se originó como un proyecto de investigación a cargo de la Agencia Nacional de Seguridad de los Estados Unidos de Norteamérica (NSA) [NSA07], [NSA09] en el año de 2003 aproximadamente, con la finalidad de proteger a sus sistemas. SELinux es una implementación de un control de acceso obligatorio flexible y de granularidad fina en el kernel de Linux. La granularidad se refiere a que tan específicos o detallados pueden ser los permisos de acceso aplicables a un objeto, lo cual para una granularidad fina se traduce en permitir una amplia variedad de operaciones diferentes sobre un mismo objeto.

3.1 Descripción

SELinux no es una distribución autónoma, sino que forma parte de las principales distribuciones de Linux. Utiliza una característica del kernel 2.6 llamada LSM (Linux Security Modules Interface).

SELinux puede cumplir una política de seguridad administrativamente definida sobre todos los procesos y objetos en el sistema, basando las decisiones en etiquetas que contienen una variedad de información de seguridad relevante. La arquitectura proveé flexibilidad mediante una clara separación de la lógica de toma de decisión de la política y la lógica de cumplimiento de la política. La lógica de toma de decisión de la política es encapsulada

dentro de un componente simple conocido como servidor de seguridad con una interface de seguridad general [Smalley05].

McCarty en [McCarty05] señala que SELinux está diseñado para proteger en contra de usos no autorizados o indebidos tales como:

- Lectura no autorizada de datos y programas.
- Modificación no autorizada de datos y programas.
- Interferencia con otros procesos.
- Escalado de privilegios.
- Violaciones de seguridad de la información
- Eludir mecanismos de seguridad de aplicación.

SELinux maneja el concepto del principio de privilegio mínimo, limita la propagación de errores y la escalación de privilegios por medio del confinamiento, así como ayudar a mantener la confidencialidad de los datos. También proveé protección de seguridad reforzada del kernel en contra de ataques como: exploits, virus y Troyanos. SELinux implementa un modelo de seguridad que es una combinación del modelo de Cumplimiento de Tipo (TE) y el modelo de Control de Acceso Basado en Roles (RBAC).[Smalley05]

3.2 Operación

SELinux tiene dos modos de operación que son: modo permisivo y modo de cumplimiento. En el modo permisivo no se cumplen las denegaciones de acceso, sino que se hace una escritura a bitácora de la denegación pero se permite que el sujeto accese a al objeto, aún cuando no tenga el permiso de hacerlo, este modo es usado durante el desarrollo de una política. En el modo de cumplimiento se hacen cumplir las denegaciones de acceso y solo se permiten los accesos que han sido especificados de forma explícita, este modo es usado durante la implementación de una política [MacAllister09].

3.2.1 Control de acceso en Linux

El control de acceso estándar de Linux utiliza para los procesos IDs de usuario (uid) y de grupo (gid) y para los objetos del sistema de archivos utiliza modos de acceso que definen los permisos sobre estos elementos (lectura, escritura y ejecución)

```
rwx r-x --- uid  gid
```

El primer grupo de permisos son para el dueño del objeto, el segundo es para el grupo y el tercero es para los que no son ni el dueño ni pertenecen al grupo (todo el mundo). Las reglas de acceso son fijas y están incorporadas en el kernel.

3.2.2 Control de acceso en SELinux

Todo control de acceso de un sistema operativo esta basado en algún tipo de atributo de control de acceso, asociado con los sujetos y los objetos. En SELinux este atributo, que en realidad es un conjunto de atributos es llamado **contexto de seguridad** [Mayer06]. Todos los objetos y sujetos tienen un solo contexto de seguridad, el cual tiene un formato: **usuario:rol:tipo:nivel** como se muestra en la Figura 3.1 , en donde *system_u* es el identificador de usuario, *system_r* el identificador de rol, *sysadm_t* el identificador de tipo y *s0* el identificador de nivel. Un contexto de seguridad válido debe tener identificadores válidos, los cuales son definidos por el desarrollador de la política. Los sujetos y objetos reciben su contexto de seguridad a partir de la política instalada. Por conveniencia y eficiencia SELinux almacena los contextos de seguridad en una tabla. Un identificador de seguridad (Security identifier -SID) identifica cada entrada de la tabla.

El contexto de seguridad es exclusivo de los atributos de seguridad en SELinux. Se ha determinado seguir la convención de agregar una terminación específica al nombre de cada atributo al momento de definirlo, como una manera de poder identificar el atributo al cual se hace referencia en un determinado momento [Strand08], esto se muestra en la Figura 3.2.

SELinux no reemplaza el DAC tradicional usado en Linux. SELinux introduce MAC en adición a DAC. Todas las decisiones de acceso son consultadas primero en DAC y después en MAC. Si una acción es denegada en DAC, MAC ya no se consulta y la acción es denegada. Si

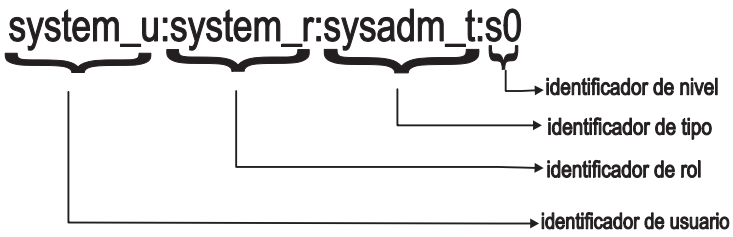

Figura 3.1: Contexto de seguridad

Atributo de seguridad	Convención de complemento de nombre	Ejemplo
Usuario	_u	system u
Rol	_r	object_r
Tipo	_t	mytype_t
Nivel	sin convención	s0

Figura 3.2: Convenciones de escritura de nombres para los atributos de seguridad.

DAC permite llevar a cabo la acción, la decisión es enviada a SELinux para una verificación MAC [Strand08]. en la Figura 3.3 se muestra que para que el sujeto pueda acceder al objeto, esto debe ser validado primero por DAC y después por MAC, si e saprobadopor ambos, entonces la acción se lleva a cabo.

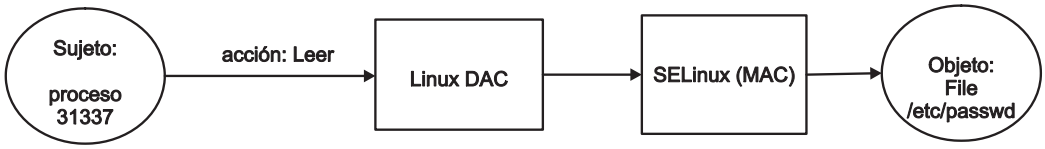

Figura 3.3: Acceso a un objeto en SELinux

En la Figura 3.4 se muestra más a detalle, los distintos componentes que conforman a MAC y que intervienen en el proceso de toma de decisión. El sujeto (proceso) desea tener acceso al objeto (que es un archivo ubicado en /etc/passwd). Este acceso debe primero ser concedido por DAC, si DAC no concede el acceso entonces termina la operación, en caso contrario se continua hacia SELinux. En SELinux el *servidor de cumplimiento de políticas*

hace una verificación en el *caché de vector de acceso* (Access Vector Cache -AVC) en el cual los permisos de los sujetos y de los objetos son alojados juntos. Si no se puede tomar una decisión basada en los datos almacenados en el AVC, la petición continua al *servidor de seguridad*, el cual verifica el contexto de seguridad del archivo y consulta la política. en la base de datos de políticas El permiso es entonces concedido o denegado (se escribe un reporte a bitácora de la denegación) y el resultado es almacenado en el AVC.

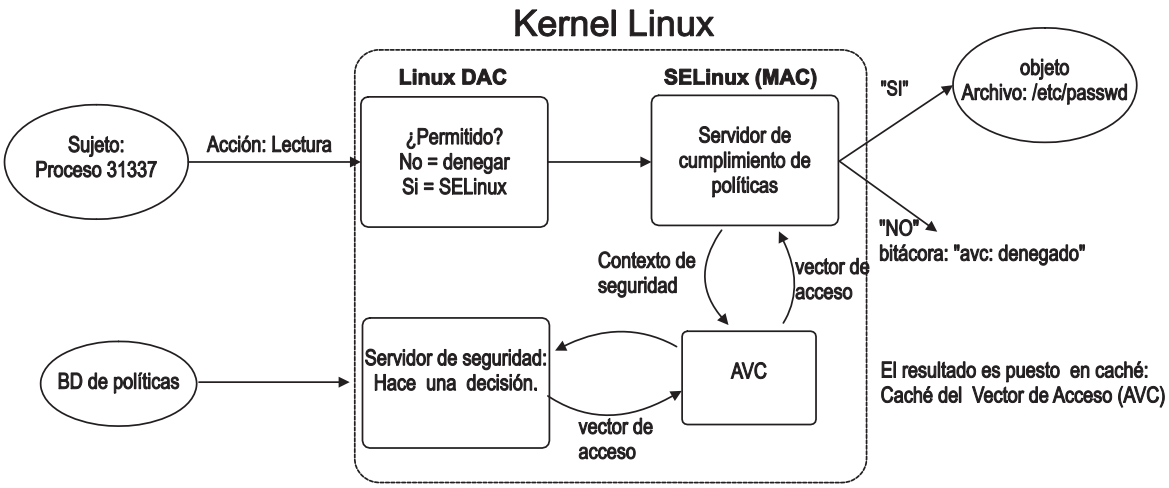

Figura 3.4: Detalle de acceso de un sujeto a un objeto.

En su mayor parte SELinux es casi transparente para la mayoría de los usuarios. Solamente los administradores del sistema se deben preocupar sobre lo estricto que debe ser una política a implementar en sus entornos de servidores. La política puede ser tan estricta o tan relajada como se requiera, y es bastante detallada. Este detalle le da al kernel SELinux un control total y granular sobre el sistema completo.

3.2.3 Clases de objetos

Mayer [Mayer06] específica que las clases de objetos y sus permisos asociados son la base del control de acceso en SELinux. Las clases de objetos representan categorías de recursos tales como archivos y sockets, y los permisos representan accesos a estos recursos tales como

lectura o envío. Entender las clases de objetos y permisos es un aspecto difícil de SELinux debido a que requieren conocimientos tanto de Linux como de SELinux. Una clase de objeto representa todos los recursos de un tipo (por ejemplo, archivos o sockets). Una instancia de una clase de objeto (por ejemplo, un archivo o socket específico) es simplemente llamado *objeto*. A menudo los términos: *clase de objeto* y *objeto* son usados indistintamente, pero es importante comprender la diferencia, clase de objeto se refiere a la categoría completa de recursos (por ejemplo, archivos); objeto se refiere a una instancia específica de la clase de objeto (por ejemplo, el archivo: passwd)

SELinux define 41 clases de objetos del kernel, cada una tiene un conjunto de acciones o permisos asociados definidos para controlar las operaciones sobre objetos con esa clase. El conjunto de permisos se representa por medio de un mapa de bits llamado: *Vector de Acceso* (Access Vector -AV), en donde cada bit se corresponde a un permiso definido para la clase. La Figura 3.5 muestra un mapa de bits simplificado para la clase File. Un mapa de bits real para esta clase incluiría una docena de acciones, pero por simplicidad se muestran las más comunes.

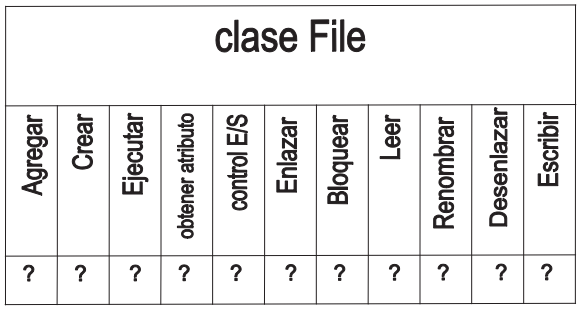

Figura 3.5: Vector de acceso.

El servidor de seguridad hace decisiones de acceso considerando los contextos de seguridad del sujeto y el objeto de la acción, la clase de seguridad del objeto, y la acción solicitada. Cuando el servidor de seguridad ha hecho la decisión de acceso, regresa un vector de acceso que indica las acciones permitidas, como se muestra en la Figura 3.6, en la cual el servidor ha concedido al sujeto los permisos de agregar al objeto o crear el objeto perteneciente a la

clase File [McCarty05].

	clase File										
	Agregar	Crear	Ejecutar	Obtener atributo	Control E/S	Enlazar	Bloquear	Leer	Renombrar	Desenlazar	Escribir
Permitir (allow)	X	X	-	-	-	-	-	-	-	-	-

Figura 3.6: Permisos de vector de acceso.

3.2.4 TE en SELinux

El Servidor de Seguridad implementa un modelo de seguridad que es una combinación de un modelo TE, un modelo RBAC y opcionalmente un modelo de seguridad multinivel (MLS) . El modelo TE proveé un control de granularidad fina sobre todos los procesos y objetos en el sistema, el modelo RBAC proveé un alto nivel de abstracción para simplificar la administración de usuarios [Smalley05].

TE es el mecanismo primario de SELinux. Es usado para especificar todos los accesos válidos entre tipos de sujetos o dominios y tipos de objetos. TE es un lenguaje de políticas de propósito general, es flexible y altamente configurable, es extremadamente granular y continúa siendo una seguridad obligatoria. TE proveé un medio para definir reglas para restringir el acceso de procesos a objetos del sistema basado en los tipos asignados a los procesos y objetos.

TE impone un control de acceso usando tipos. Así, el primer paso para cualquier política es la de definir tipos. Los tipos son las piezas centrales del cumplimiento de tipo. Un tipo es un identificador sin ambigüedad que es aplicado a todos los sujetos y objetos de la política en cuestión, y se utilizará para todas las decisiones de acceso subsecuentes (reglas TE, contextos y sentencias de etiquetado) . Todo objeto y/o sujeto que tenga el mismo tipo tiene los mismos permisos de acceso. Un tipo se define por medio de una sentencia *type,* como se muestra en

la siguiente sentencia en donde se declara un tipo de nombre *myapp_t*

```
type myapp_t;
```

Los tipos declarados son usados posteriormente en las **reglas allow** de TE , las cuales son las reglas del *vector de acceso* que sirven para especificar las interacciones permitidas entre los tipos (típicamente entre un tipo de dominio y un tipo de objeto), es decir, son las que especifican los permisos de acceso para las clases de objetos; esto es necesario, ya que ningún acceso es llevado a cabo sin la asociación de una regla allow. Un permiso asocia un tipo, una clase y un conjunto de operaciones (ver Figura 3.7).

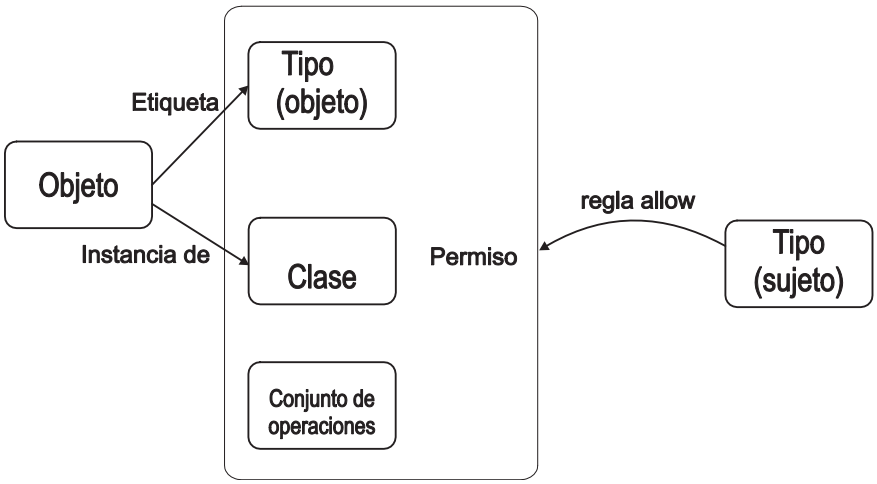

Figura 3.7: Permiso en TE.

La sintaxis de una regla allow es como sigue:

```
allow tipo_fuente tipo_destino : (clases de objetos) permisos;
```

Donde:

tipo_fuente, son los sujetos o dominios.

tipo_ destino, son los objetos a los cuales se va a permitir el acceso.

clases de objetos, son las clases para las cuales aplica el acceso.

permisos, son los permisos especificos otorgados.

Ejemplo:

```
allow user_t bin_t : file {read execute write};
```

Este ejemplo muestra la sintaxis básica de una regla allow de TE, esta regla tiene dos tipos de identificadores: *user_t* que es el tipo fuente (o sujeto o dominio); *bin_t* que es tipo objetivo (u objeto). El identificador *file* es el nombre de la clase de objeto definido en la política (en este caso representa un archivo ordinario). Los permisos *read*, *execute* y *write* contenidos dentro de los parentesis son un subconjunto de permisos válidos para una instancia de la clase de objeto File. La traducción de esta regla sería: Un proceso con un tipo dominio user_t puede leer, ejecutar y escribir un objeto File con un tipo bin_t. (ver Figura 3.8).

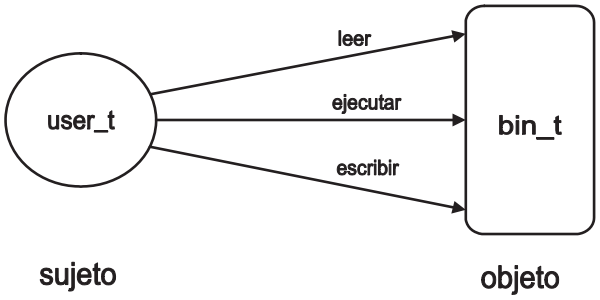

Figura 3.8: Regla allow.

Otro ejemplo sería:

```
allow user_t passwd_exec_t: file { getattr execute};
```

Esta regla permite que los procesos con un tipo dominio *user_t* puedan obtener los atributos y ejecutar el programa *passwd*. la Figura 3.9 muestra como es que por medio de esta regla allow se permite que un usuario que ingresa al sistema (login), pueda con su uid y su tipo ejecutar el shell bash, y posteriormente hace un fork() para ejecutar el programa de passwd que hace uso del comando execv(), pero para que este comando se pueda ejecutar sobre passwd, es necesario que el tipo del usuario lo tenga permitido, entonces eso es lo que esta regla allow esta haciendo, que un sujeto con un tipo user_t pueda abtener atributos y ejecutar un objeto passwd_exec_t que pertenece a la clase file.

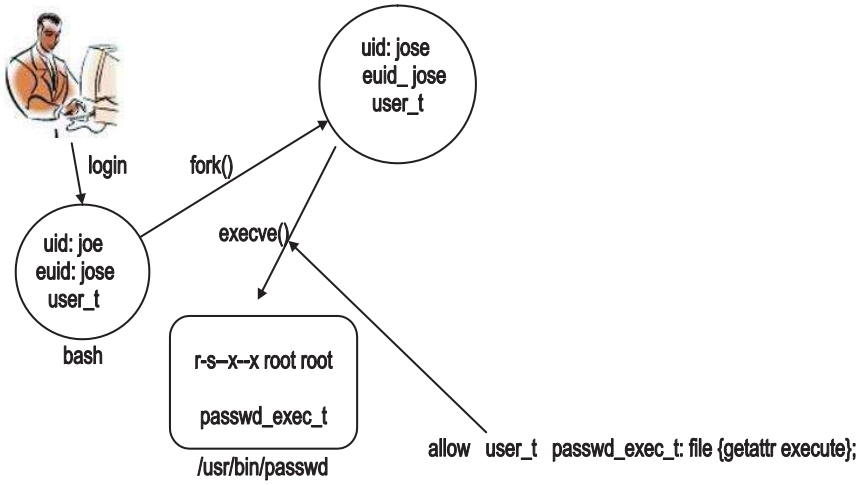

Figura 3.9: Ejemplo de utilización de regla allow.

Como ya se mencionó anteriormente SELinux hace uso del control de acceso obligatorio para reforzar el privilegio mínimo en los procesos. La idea es que si un proceso que está siendo ejecutado es comprometido por un intruso, éste no sea capaz de obtener privilegios extra o elevarlos y de esa forma extender el daño a otras partes del sistema; TE se encarga de evitar que esto suceda, ya que le proporciona al kernel una protección denominada *cajas de arena* o *sandboxes*, las cuales permiten confinar al proceso proporcionandole solo lo que necesita para hacer su tarea, y en caso de que este llegué a fallar o ser comprometido, este daño quede limitado en su caja de arena. (ver Figura 3.10 tomada de [Strand08]).
La meta primordial de una política TE es la de autoprotección, esto es, el kernel se protege a sí mismo y a sus recursos.

Una diferencia entre el modelo SELinux TE y el modelo tradicional es que el modelo SELinux TE no asocia directamente a los usuarios con los dominios, en lugar de eso, SELinux utiliza el modelo RBAC para proveer una capa adicional de abstracción entre los usuarios y los dominios [Smalley05].

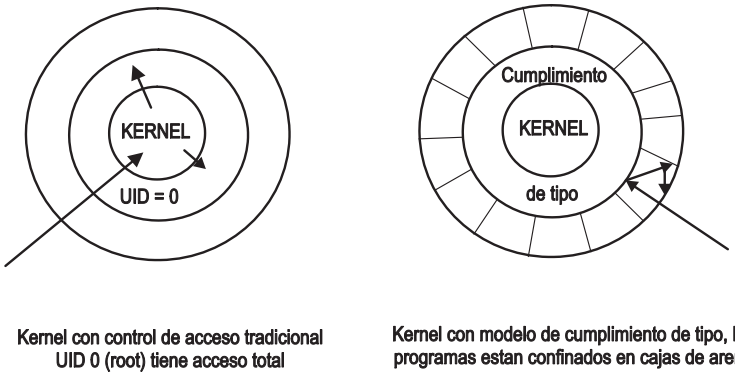

Figura 3.10: Comparación de un kernel con control de acceso tradicional y un kernel con cumplimiento de tipo.

3.2.5 RBAC en SELinux

Un modelo RBAC tradicional autoriza a usuarios a actuar en ciertos roles, y asigna un conjunto de permisos para cada rol. El modelo RBAC SELinux autoriza a cada usuario para un conjunto de roles, y autoriza a cada rol para un conjunto de dominios TE [Petersen08]. Una relación de dominio de roles puede opcionalmente ser especificada en la configuración RBAC para definir una jerarquía entre roles. Esta aproximación combina la facilidad de administración provista por el modelo RBAC con las protecciones de granularidad provistas por el modelo de TE [NSA07].

La característica RBAC de SELinux esta soportada sobre TE; el control de acceso en SELinux es fundamentalmente vía TE. Los roles limitan los tipos a los cuales un proceso puede transitar basandose en el identificador de rol en el contexto de seguridad del proceso. De esta manera, el desarrollador de una política puede crear un rol que tenga permitido transitar dentro de un conjunto de dominios (asumiendo que las reglas de TE permiten la transición), definiendo de ese modo los limites del rol [Mayer06]. Los roles primarios definidos en SELinux son: system_r (para procesos de sistema), sysadm_r (procesos de usuario administrativo, no necesariamente el nombre de usuario “root”), y user_r (procesos de usuario no privilegiado).

El modelo RBAC SELinux mantiene un atributo de rol en el contexto de seguridad de cada proceso. Para objetos, el atributo de rol es típicamente fijado a un rol genérico:

object_r.
Lo que hace RBAC en SELinux :

- Asignar privilegios y autorizaciones a roles.
- Autorizar usuarios para uno o más roles.
- Asignar privilegios y autorizaciones a tipos de dominio.
- Asignar tipos de dominio a roles.
- Soportar el concepto de jerarquía de roles.

La sentencia de declaración de rol define un rol y los tipos de dominio en los cuales los procesos en ese rol pueden hacer una transición. La sintaxis general de una declaración de rol es:

```
role nombre_rol types tipos_ permitidos;
```

En donde, el campo *nombre_ rol* es el único nombre para el rol; el campo *tipos_ permitidos* debe incluir uno o más tipos y/o atributos indicando el tipo de dominio para el cual un proceso en ese rol puede hacer una transición.
ejemplo :

```
role user_r types passwd_t;
```

Esta regla le permite al rol user_r hacer transición al tipo passwd_t. La Figura 3.11 continua con el ejemplo de la regla allow, en donde fue necesario agregar la regla que explicitamente le permitiera al usuario con un tipo user_t poder ejecutar un objeto del tipo passwd_exec_t, pero eso no es suficiente para la ejecución dado el programa passwd requiere que el tipo sea passw_t, se debe asignar ese tipo al rol del usuario por medio de la sentencia arriba mencionada.

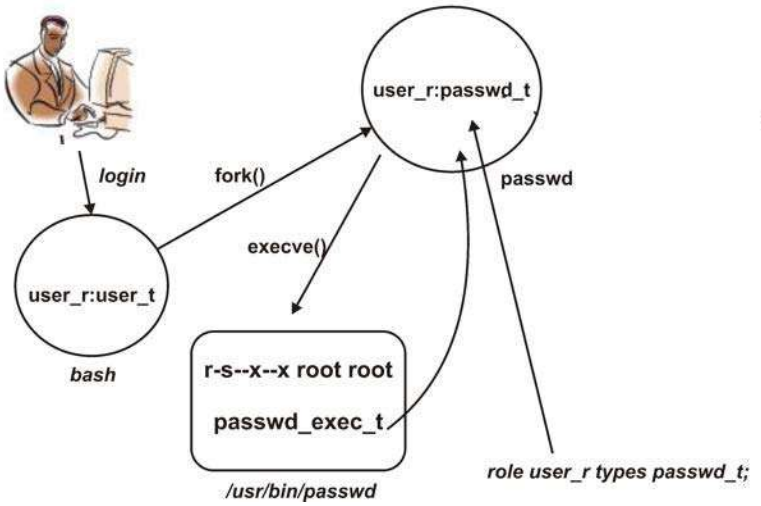

Figura 3.11: Ejemplo de autorización de rol.

3.2.6 Macros de permisos

Las macros proveen abstracciones fáciles de usar. Pueden ser usadas en lugar de una lista explicita de permisos, por ejemplo para sustituir un conjunto de reglas allow.
Ejemplos de macros de permisos:

r_file_perms (leer archivos y sus atributos)

permisos: *read, getattr, lock,* y *ioctl*

r_dir_perms (leer directorios y sus atributos)

permisos: *read, getattr, lock, search,* y *ioctl*

rw_file_perms (leer y escribir archivos y sus atributos)

permisos: *read, getattr, lock, write* y *ioctl*

3.2.7 Comandos SELinux

El desarrollo de este proyecto se llevó a cabo sobre la plataforma Linux con la distribución Fedora [Negus08] [Red Hat07],[FedoraForum08] en su versión número 9 que utiliza el kernel 2.6.x. Una de las principales razones para trabajar con esta distribución de Linux, es el hecho

de que Fedora ha venido incluyendo a SELinux como parte de su distribución, integrando los módulos de seguridad a su kernel. Se puede decir que ha ido ganando experiencia en la interacción de Linux y SELinux, como son: la modificación de algunos de los comandos de Linux para que actúen de forma similar a como lo hacen, pero con características extendidas para su uso en SELinux y la inclusión de nuevos comandos, creados especialmente para su uso en SELinux. Éstas son algunas características que hacen que el manejo de SELinux sea más clara y estable ya que permiten un mejor control y facilidades para la creación, modificación y administración de las políticas, además de documentación disponible a través de la página de Fedora y el proyecto SELinux [SELinux07] y la página de la NSA dedicada a SELinux [NSA09].

Un ejemplo de un comando modificado es el comando *ls*, cuya función normal en Linux es la de mostrar un listado con los archivos y directorios, en donde para cada archivo o directorio muestra sus permisos de acceso y su id de usuario y su id de grupo; este comando usado en SELinux con la opción *-Z* , además de mostrar la información antes mencionada, desplegará el contexto de seguridad del mismo. Por ejemplo usando este comando en un archivo llamado: *expediente20* el cual pertenece al usuario *fdiaz*, al grupo *despacho* y que haya sido etiquetado con el contexto de seguridad: *unconfined_u:object_r:user_home_t:s0* mostrará la siguiente salida:

```
# ls  -Z   expediente20
# -rw-rw-r-- fdiaz despacho unconfined_u:object_r:user_home_t:s0 expediente20
```

Además de los comandos modificados, SELinux cuenta con sus propios comandos, algunos de los más importantes son:

- **Semodule**. Maneja la instalación, actualización (upgrading) y remoción de módulos de políticas cargables (loadable policy modules).
- **Semanage**. Maneja la adición, modificación y remoción de usuarios, roles, archivos de contexto, etiquetado de puertos y etiquetado de interfaces [MacAllister09]
- **Restorecon**. Este programa es usado principalmente para establecer el contexto de seguridad de uno o más archivos.

3.3 Conclusiones

En este capítulo se hizo una descripción de lo que es SELinux; se habló de cómo está integrado al kernel de Linux junto con los demás elemento para poder llevar a cabo su función de proveer el control de acceso obligatorio. Se ha comparado con el control de acceso tradicional de Linux y la forma en que lo complementa, reforzando la seguridad del sistema por medio de la utilización de políticas, los cuales están escritos en su propio lenguaje de políticas para especificar que es lo que se le permitirá hacer a la entidad a la que se le aplique.

Como se mencionó, SELinux hace uso del principio de privilegio mínimo para que las aplicaciones lleven a cabo su función haciendo uso de solo aquellos recursos necesarios, evitando de esa forma riesgos al sistema que se pueden dar por medio de la escalación de privilegios, tal como sucede cuando solo se usa DAC, en donde un solo usuario que cuenta con todos los privilegios que usados de forma incorrecta o malintencionada pueden provocar daños no solo a la aplicación sino a todo el sistema.

Capítulo 4

Arquitectura Cliente-Servidor

En este capítulo se explica que es la arquitectura Ciente-Servidor, así como los elementos que la conforman y la manera en que se lleva a cabo la comunicación entre el cliente y el servidor. También se explica el funcionamiento de IPtables.

4.1 Sockets

Designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada. Un socket queda definido por una dirección IP, un protocolo de transporte y un número de puerto.

4.2 Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor consiste en una conexión que se establece entre una máquina cliente que realiza peticiones y una máquina servidor que responde a ellas. La arquitectura funciona tanto local como remotamente, por lo que cliente y servidor pueden residir en la misma computadora y desde allí llevar a cabo la conexión o bien estar cada uno en distintas partes de una red y también llevar a cabo la conexión; por lo tanto la separación entre ellos es de tipo lógico.

El servidor puede ofrecer distintos servicios a los clientes dependiendo para que haya sido diseñado. Además de poder atender a varios clientes a la vez. El servidor se puede localizar en cualquier nodo de la red y gestionar el acceso a un determinado recurso. El servidor atiende a los clientes a través de un puerto en concreto que está constantemente escuchando las solicitudes de los clientes que se conectan a él.

El cliente realiza peticiones de servicio al servidor, espera y recibe respuesta de él. El cliente puede conectarse a varios servidores. La mayoría de las veces el cliente es el que interactúa con los usuarios finales. La Figura 4.1 muestra de manera muy sencilla la interacción Cliente-Servidor: el cliente hace una solicitud de servicio al servidor, este la procesa y le envía la respuesta solicitada.

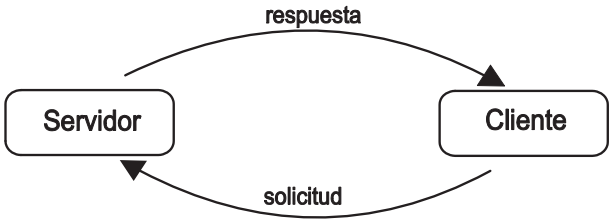

Figura 4.1: Interacción cliente y servidor.

Ejemplos de Cliente-Servidor son: Servidor web, servidores de bases de datos, servidor de correo, servidor de impresión, etc. En el servidor web el cliente sería el dispositivo que puede observar el vídeo, imagenes y texto, o reproducir el audio distribuido, elementos que son proporcionados por el servidor.

La arquitectura Cliente-Servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico. En la arquitectura monolítica todo es llevado a cabo por el kernel que es grande y complejo, y engloba a todos los servicios del sistema por lo que este se ocupa de atender todas las solicitudes de servicios del sistema; en cambio, en la arquitectura Cliente-Servidor se libera al kernel de tanta carga de trabajo por medio del uso de servidores que atienden tareas especificas. Ejemplos de aplicaciones que trabajan bajo la arquitectura Cliente-Servidor son: servidores FTP, donde las aplicaciones

más comunes de los servidores FTP suelen ser el alojamiento web, en el que sus clientes utilizan el servicio para subir sus páginas web y sus archivos correspondientes; un servidor de respaldo de archivos, que hará copias de seguridad de los archivos importantes de una empresa; un servidor de bases de datos que envia a sus clientes los registros que le solicitaron a través de manejador de bases de datos.

En la Figura 4.2 se muestra como se establece la comunicación entre un cliente y un servidor por medio de sockets. Se tiene al cliente y al servidor, ambos realizan unas serie de funciones para poder estalecer comunicación y llevar a cabo el proceso de solicitud- respuesta, Se describe primero la serie de llamadas del lado del servidor y después las del lado del cliente. El servidor usa la siguiente secuencia de llamadas a funciones:

1. Llamada a la función *socket( )*, con la que se obtiene un descriptor de socket, en el cual se identifica entre otras cosas el protocolo de red que se usara para la comunicación, por ejemplo: TCP

2. Función *bind( )* la cual le da un nombre al socket para que pueda recibir las conexiones. Es con esta función con la que se proporciona una dirección IP y un número de puerto.

3. La función *listen( )* le permite al servidor aceptar conexiones entrantes de los clientes (ponerlos en cola), en esta función se especifica el número máximo de conexiones entrantes (tamaño de la cola).

4. El servidor usa la función *accept( )* para aceptar una petición de conexión entrante. Esta llamada mantiene al servidor bloqueado en espera hasta que se reciba la conexión de un cliente.

5. Con la función *read( )* el servidor recibe datos del cliente (petición).

6. La función *write( )* envia datos al cliente (respuesta)

7. *close( )* cierra el socket. Libera la conexión.

El cliente usa la siguiente secuencia de llamadas a funciones:

1. Llamada a la función *socket( )* con la que se obtiene el descriptor de socket para el cliente.

2.*connect( )* establece una conexión con el servidor.

3.La función *write( )* envia datos al cliente (petición).

4.*read( )* lee lo que el servidor le envía (respuesta).

5. *close( )* cierra el socket. Libera la conexión.

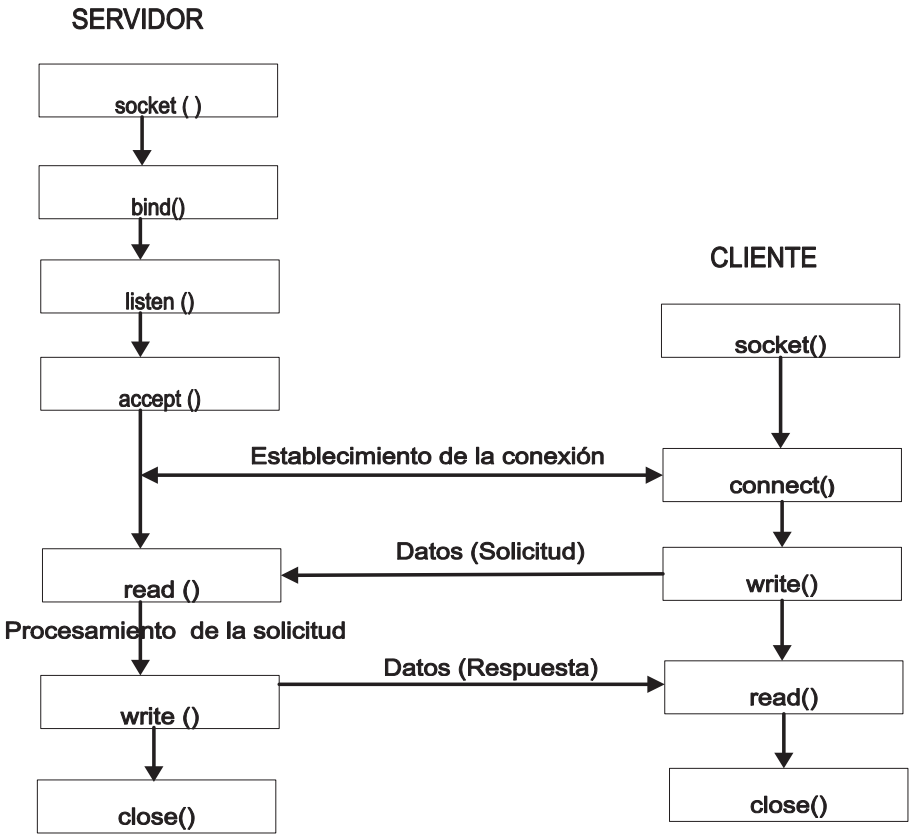

Figura 4.2: Comunicación entre cliente y servidor.

4.3 IPtables

IPtables [Rash07], [Purdy04] es un firewall o cortafuegos vinculado al kernel de Linux que se ha extendido enormemente a partir del kernel 2.4 de este sistema operativo. IPtables está integrado con el kernel, es parte del sistema operativo. IPtables se basa en la aplicación de reglas de filtrado de red con las que se decide si una conexión determinada se establece o no.

Su forma de operar es que cuando un paquete llega a la red, éste es inspeccionado por IPtables, el paquete pasa a través de una secuencia de reglas de filtrado para su procesamiento. Las reglas se agrupan en cadenas y las cadenas se agrupan en tablas, donde cada tabla está asociada con un tipo diferente de procesamiento de paquetes. Hay un total de 3 tablas. La primera es la MANGLE, la cual es la responsable de la alteración de los bits de calidad de servicio en la cabecera TCP. La segunda tabla es la FILTER, la cual es responsable del filtrado de paquetes, esta tabla tiene 3 cadenas fijas en las cuales uno puede poner las reglas de política para el firewall, y son:

FORWARD Esta cadena se usa para filtrar todos los paquetes que no se generan localmente y que no están destinados a nuestro firewall.

INPUT Esta cadena se emplea para filtrar todos los paquetes que se destinan a nuestro firewall.

OUTPUT Esta cadena se emplea filtrar todos los paquetes generados localmente.

La tercera tabla es la de NAT la cual es responsable de la tradución de direcciones de red. Esta tabla tiene 2 cadenas fijas que son:

PRE-ROUTING Esta cadena realiza la traducción de direcciones cuando la dirección destino del paquete necesita ser cambiada.

POST-ROUTING Esta cadena realiza la traducción de direcciones cuando la dirección fuente del paquete necesita ser cambiada.

Cada regla del firewall inspecciona cada paquete IP y trata de identificarlo como el objetivo de alguna clase de operación. Una vez que un objetivo es identificado, el paquete se dirige hacia él para su procesamiento. El destino de una regla puede ser el nombre de una cadena definida por el usuario o uno de los destinos ya fijos que IPtables utiliza, los cuales son:

ACCEPT El paquete es aceptado y transferido a la aplicación final o al sistema operativo para su procesamiento.

DROP El paquete es bloqueado sin notificación para la fuente que lo envía.

La estructura general de una regla iptables es:

```
iptables -t [tabla] -[ALFPDR][cadena]  [criterio de concordancia] -j [cadena]
```

donde:

-t [tabla]. Especifica a cual de las tablas se añadirá la regla, según se especifique en el campo *tabla*. La tabla por defecto es la de filtrado.

-[ALFPXZ][cadena]. Cada una de estas opciones se aplican sobre la cadena especificada en el campo *cadena*.

A. Añade una regla al final de la cadena.

L. Lista las reglas de la cadena.

F. Borra las reglas de la cadena, si no se especifica la cadena, se borran las reglas de todas las cadenas.

P. Configura la política por defecto para una cadena en particular, de tal forma que, cuando los paquetes atraviesen la cadena completa sin cumplir ninguna regla, serán enviados a un objetivo. en particular, como puedan ser ACCEPT o DROP.

X. Borra la cadea de la tabla.

Z. Pone a cero los contadores de la cadena.

[criterio de concordancia]. Especifica las características que debe tener el paquete para coincidir con esta regla.

-p [tipo_protocolo]. Especifica el protocolo del paquete. Los mas comunmente usados son: TCP, UDP e ICMP.

-s [dir_ip]. Especifica la dirección IP fuente del paquete.

-d [dir_ip]. Especifica la dirección IP destino para el paquete.

-i [nombre_interfaz]. Especifica el nombre de la interfaz de red por la cual ingresa el paquete.

-o [nombre_interfaz]. Especifica el nombre de la interfaz de red por la cual sale el paquete.

-sport [numero_puerto]. Especifica el puerto origen del paquete.

-dport [numero_puerto]. Especifica el puerto destino del paquete.

-j [cadena] Se decide que se hace con el paquete que coincide con la regla . Las opciones básicas son: aceptarlo (ACCEPT) o rechazarlo (DROP).

Ejemplos:

```
iptables -A INPUT -s 192.168.1.10 -d 10.1.15.1 -p tcp
--dport 22 -j ACCEPT
```

Lo que se está haciendo en este primer ejemplo es permitir el tráfico de paquetes de red entre la dirección fuente IP 192.168.1.10, a la dirección destino IP 10.1.15.1. a través del puerto 22

```
iptables -A INPUT -p tcp -i eth0 --dport 80 -j DROP
```

En este segundo ejemplo, la regla va a rechazar todos los paquetes TCP que entren por la interfaz eth0 que vayan dirigidas hacia el puerto 80 (HTTP).

En la Figura 4.3 se muestra a IPtables aplicando el filtrado de red a los paquetes que examina, así si el paquete es de origen remoto y destino local (son los paquetes que

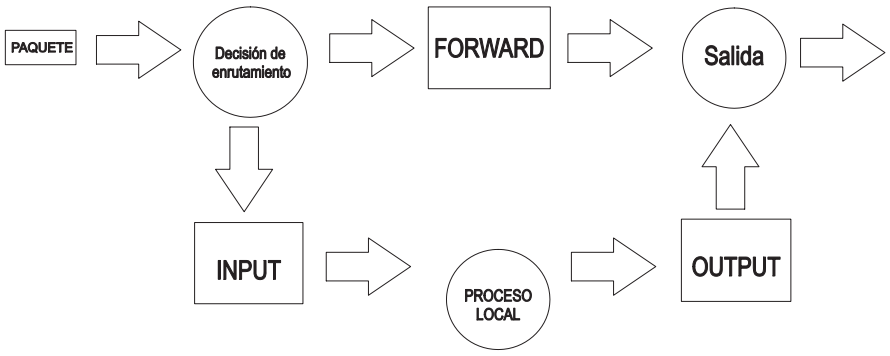

Figura 4.3: Proceso de filtrado de paquetes con IPtables

las máquinas de cualquier red envían a la nuestra), se analiza mediante el filtro de entrada INPUT; si el paquete es de origen remoto y destino remoto (son los paquetes, que actuando como gateway, nuestra máquina tiene que reenviar), se analiza mediante el filtro de reenvío FORWARD y si el paquete es de origen local y destino remoto (paquetes que nuestra máquina envía a otras máquinas) se analiza mediante el filtro de salida OUTPUT.

4.4 Conclusiones

En este capítulo se consideró importante hacer la descripción de la arquitectura Cliente-Servidor, dado que sobre ella se hará la implementación de los controles de acceso obligatorios mediante la interfaz desarrollada (CS-Enhanced). Tal como se mencionó en este capítulo, la arquitectura Cliente-Servidor consta de dos elementos principales que son: el cliente y el servidor, además del puerto de red que se define como el canal de comunicación entre ambas entidades. Las aplicaciones creadas bajo esta arquitectura normalmente llevan a cabo la función para la cual fueron diseñadas, intercambiando mensajes con información entre el servidor y los clientes. Se mencionaron ejemplos de servidores que se usan de manera cotidiana en los sistemas computacionales, como los servidores web y los servidores de bases de datos. Todo hasta ese momento no presenta ningún problema en su funcionamiento,el cliente pide un servicio y el servidor se lo ofrece, y así mientras este tenga conexiones disponibles para

aceptar a los clientes que se conectan a su puerto, pero queda la pregunta de ¿Qué es lo que pasa con la seguridad? ¿cómo este tipo de aplicaciones se protegen de usos inadecuados?o ¿Comó evitar que sean usadas por otra aplicación para escalar privilegios? Este tipo de preguntas van a ser resueltas por la interfaz desarrollada cuyo objetivo es el de implementar controles de acceso obligatorios y filtrado de red como se verá en el capítulo siguiente.

Capítulo 5

CS-Enhanced

Este capítulo está dedicado a describir la interfaz que desarrollado y que denominamos CS-Enhancement o simplemente CS-E. Esta interfaz es una herramienta de software por medio de la cual se implementa el control de acceso obligatorio en una arquitectura Cliente-Servidor. Se describirá la estructura, el funcionamiento y la interacción de los distintos elementos de software que conforman dicha herramienta. Se incluye además un ejemplo real de la aplicación de CS-E.

El desarrollo de CS-E se llevó a cabo por medio del lenguaje Perl [Christiansen98], [Schwartz97], [Srinivasan97], [Wall96], [Siever98], [Randal03], [Perl09], [Wall00], se utilizó un módulo especial de Perl llamado:Parse::RecDescent de CPAN para el manejo de expresiones regulares en una grámatica [Orwant02], IPtables [Netfilter09] y SELinux [SELinux07] que viene incluido en el sistema operativo Fedora [Negus08], [Red Hat07], [FedoraForum08], que fué la distribución de Linux que se utilizó, en este caso la versión 9, que utiliza el kernel 2.6.x. y que incluye a SELinux.

Tal como se mencionó en el capítulo anterior, la arquitectura Cliente-Servidor consta de una parte que es el servidor, el cual presta un servicio a la otra parte que son los clientes que se conectan a él, atendiendo todas las peticiones que le lleguen, sin ninguna restricción, más que la del número de conexiones máxima especificado, así que básicamente cualquier cliente de la red que conozca la dirección IP del servidor y el puerto en el éste está escuchando,

puede conectarse a él y solicitarle el servicio que proveé. El objetivo de CS-E es implementar un control de acceso que asegure que quien opera al cliente y al servidor esté autorizado para hacerlo y de forma en que tiene autorizado hacerlo, además de restringir el uso del puerto a otras aplicaciones. Para poder cumplir hacer cumplir los puntos que se acaban de mencionar, el control de acceso tiene que ser obligatorio .

La implementación se lográ creando una serie de archivos de bajo nivel en lenguaje de políticas de SELinux y un script de IPtables. Las reglas que se generan para el script, estarán vinculadas a la política SELinux desarrollada, de modo que el filtrado de paquetes de red estará fuertemente ligado y acorde con las reglas de acceso definidas en SELinux.

5.1 La relevancia de CS-E

En distintas partes de este trabajo hemos hablado acerca de la importancia de la información dentro de las organizaciones como uno de los aspectos más importantes a tener en cuanta, dado que la información constituye un activo de vital importancia para el funcionamiento de una organización. Las organizaciones basan sus operaciones con base en la información que reciben, generan, y procesan. La información debe mantenerse protegida de usos inadecuados. Se debe tratar de mantener en todo momento las propiedades de confidencialidad, integridad y disponibilidad de la información, tal como se mencionó al principio del capítulo 2.

No obstante de tener presente la importancia que tiene la información dentro de una organización, y las posibles amenazas a las que se ve expuesta, muchas veces no se toman en cuenta las medidas necesarias para su protección por diversas razones, algunas de ellas son:

La dirección ejecutiva no lo considera un asunto prioritario. Esto puede llegar a suceder porque se piensa que el asunto de la seguridad de la información está resuelto por las aplicaciones que hacen uso de ellas. Es decir, que se confía en que éstas son por sí mismas lo suficientemente seguras y confiables, y no representan ningún riesgo para la seguridad de la información de la que hacen uso. Unido a lo anterior, se tiene que, generalmente en las organizaciones no existe un departamento de sistemas, ni siquiera un administrador de la red,

que se encargue del mantenimiento y el soporte; la mayoría de las veces, porque se le ve como personal innecesario que generará un gasto adicional para una cuestión que no es prioritaria; por lo tanto se permite que la seguridad de la información quede a cargo de las aplicaciones que se utilizan, de la manera en que cada una de ellas usen la información y la forma en que permitan a los usuarios hacer uso de ésta.

La cantidad de tareas por realizar. Inclusive si la organización cuenta con un administrador de la red, éste con frecuencia se encuentra todo el tiempo resolviendo una cantidad importante de tareas relacionadas a satisfacer el correcto funcionamiento de la red. Estas tareas pueden ser: configuración de servidores, soporte técnico a usuarios, configuración de dispositivos de red, creación de respaldos de información, etc, por nombrar algunas; estas tareas le demandan una gran cantidad de tiempo y atención, no dejando oportunidad para la cuestión de la seguridad de la información.

El tiempo que demora la configuración de una solución de seguridad. Suponiendo que dentro del itinerario de tareas que tiene un administrador de red, se encuentre agendado un espacio de tiempo destinado a poder trabajar con la seguridad de las aplicaciones y la información, éste generalmente es límitado. Si el administrador de la red está realmente preocupado por la seguridad, buscará optimizar este tiempo para lograr implementar algún esquema de seguridad que resulte efectivo y práctico a las necesidades de la organización, de manera que cumpla con el objetivo dentro del tiempo que ha destinado a esta tarea. Una implementación de seguridad podría tomarle un tiempo significativo de desarrollo y/o configuración, el cual dependerá de su complejidad, los conocimientos técnicos que posea y su experiencia. Un administrador tenderá a buscar una solución entre los recursos que tenga a su disposición y sea capaz de operar, optando muchas veces por aplicar aquellas soluciones que no demanden demasiado tiempo en su desarrollo y/o configuración, aunque la elección no sea muy adecuada.

CS-E busca poder resolver gran parte de esta problemática de seguridad de una manera sencilla y efectiva, enfocándose en las aplicaciones que trabajan bajo la arquitectura Cliente-Servidor. Por medio de CS-E se pone a disposición del usuario una herramienta computacional que le apoyará en la tarea de proveer seguridad a sus aplicaciones Cliente-Servidor.

La interfaz desarrollada tiene como finalidad encargarse del trabajo pesado de la implementación del control de acceso obligatorio, que consiste en definir las políticas del cliente y el servidor en el lenguaje SELinux, esto implica crear un par de archivos que contienen: definiciones de tipos, especificaciones de permisos de acceso (por medio de las reglas de acceso o reglas allow), etiquetado de archivos y etiquetado de puerto principalmente, así como la definición de cada una de las reglas de filtrado de red en IPtables, para especificar claramente cual es el tráfico que se va a permitir tener entre el cliente y el servidor para el cual se quiere hacer la implementación de MAC.

CS-E le dará a las aplicaciones que trabajan bajo la arquitectura Cliente-Servidor, protección en contra de usos inadecuados y evitará que algún fallo en su diseño pueda ser aprovechado para lograr infiltrarse al sistema y causar algún daño. Esto es posible gracias a que por medio de MAC se confina a la aplicación y se le proveé un entorno seguro para su ejecución.

La relevancia de CS-E, es que se está simplificando para el usuario un proceso que no es sencillo de llevar a cabo, porque implica la configuración de una serie de elementos, y la simplificación llega al grado de que no es necesario que el usuario conozca los detalles internos más que solo el hecho de aprender a especificar la política de seguridad en el lenguaje de alto nivel que se le presenta.

Los componentes que integrar a CS-E son:

1. Lenguaje de especificación de alto nivel. En este lenguaje se hace la especificación de alto nivel de la política de seguridad para la aplicación Cliente-Servidor.

2. Analizador sintáctico o parser para la especificación de alto nivel. Es el que hace el análisis sintáctico de la especificación de alto nivel .

3. Proceso de creación de archivos de políticas SELinux. En este proceso se crean los archivos para la política del cliente y para la política del servidor respectivamente.

4. Proceso de creación de scripts de instalación y compilación de políticas. Estos scripts contiene comandos SELinux y de Linux para la compilación y la instalación de

la política del cliente y la política del servidor.

5. Proceso de creación de script de configuración de firewall. Este script contiene las reglas de filtrado que se aplicarán a la comunicación de la aplicación Cliente-Servidor.

6. Proceso de compilación e instalación de política SELinux. En este proceso se ejecuta el script de instalación y compilación de políticas del cliente y el del servidor.

7. Proceso de etiquetado de archivos y puerto. Se etiquetan los archivos de la aplicación Cliente-Servidor con el script de instalación y compilación de políticas.

8. Proceso de puesta en marcha del firewall. Se ejecuta el script de configuración del firewall para ponerlo en funcionamiento.

En el diagrama de flujo en la Figura 5.1, se presenta gráficamente la secuencia de operación de CS-E. Se describen sus pasos:

El primer paso que realiza CS-E es el de recibir como entrada de datos a la especificación hecha en el lenguaje de alto nivel, de la política de seguridad para la aplicación Cliente-Servidor.

En el siguiente paso hace el análisis sintáctico de la especificación, la cual es guiada por una gramática libre de contexto, cuyas reglas identifican por medio de expresiones regulares a cada uno de los elementos presentes en la especificación de la política de seguridad y lo traducirán a su correspondiente implementación en un archivo de bajo nivel.

El siguiente paso verifica que los valores sean correctos, si es así se continúa, de lo contrario algo está mal en la especificación y hay que regresar a corregirla.

El paso siguiente corresponde al proceso de creación de los archivos de SELinux.

El siguiente paso es el de la creación de un par de scripts de compilación, estos scripts contienen comandos SELinux para la compilación y la instalación de la política del cliente y la política del servidor.

Paso siguiente es el proceso de creación del script del firewall.

Paso final es la ejecución del firewall.

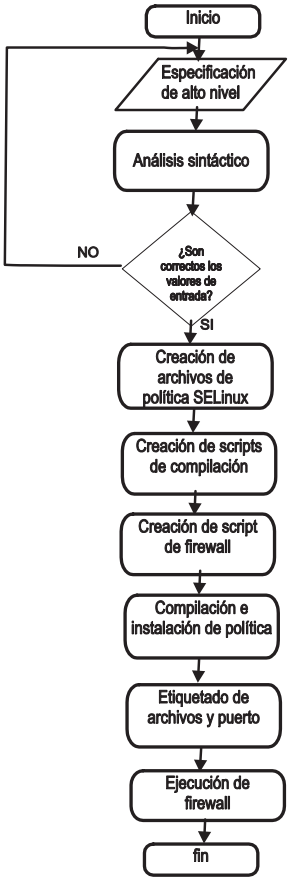

Figura 5.1: Diagrama de flujo de CS-E

5.1.1 Lenguaje de especificación de alto nivel

Como parte inicial del proyecto, implementamos un lenguaje de alto nivel definido por el Dr. Juan Manuel Garcia en [Garcia09]. A través de este lenguaje de especificación se le permite al usuario definir la política de seguridad para el cliente y el servidor. Esta especificación es la base para el proceso de creación de los archivos y scripts necesarios para la implementación del control de acceso obligatorio para la aplicación Cliente-Servidor en cuestión.

La declaración principal del lenguaje es la que define un proceso. Como se muestra en la Figura 5.2, el proceso *proc* se declara seguido de un conjunto de sentencias: sentencia1,

sentencia2...sentenciaN, que asignan valores a elementos fundamentales de la política que luego serán usados en todo el proceso de implementación que hace CS-E.

```
process proc
    {
      sentencia 1
       sentencia 2
            .
            .
            .

       sentencia N
     }
```

Figura 5.2: Sintaxis para la definición de una política de seguridad en el lenguaje de alto nivel

En donde:

process: Es una palabra reservada para indicar el inicio de un proceso. La estructura process se aplica tanto para definir los parámetros del cliente como los del servidor, por lo tanto debe existir una estructura por cada uno.

proc: variable que define el nombre del proceso para el cual se van a asignar sus parámetros, este nombre será usado para definir múltiples parámetros dentro de los archivos de la política.

sentencia: Son cada una de las líneas en donde son asignados los valores de los parámetros para el proceso.

Ejemplo de una definición de política de seguridad en el lenguaje de alto nivel para el cliente y el servidor de una aplicación Cliente-Servidor

process cliente1

{

ruta=/root/Desktop/MyClient

```
 version=1.0.0
 nombre_aplicacion=servidor
}

process servidor1
{
 ruta=/root/Desktop/MyServer
 version=1.0.0
 num_puerto= 8000
 nombre_aplicacion=servidor
 dir_ip=192.168.0.100
}
```

En este ejemplo de definición de política se muestra como de acuerdo a la sintaxis definida, se asignan nombres a las estructuras para los procesos cliente y servidor, siendo *cliente1* el nombre para el proceso cliente y *servidor1* el nombre para el proceso servidor, además del listado de asignaciones de valores que requiere la política. A continuación se describe cada uno de los elementos del cuerpo de los procesos.

Para el proceso cliente:

ruta	Especifica la ruta en donde serán creados los archivos.
version	Es el número de versión de la política.
nombre_aplicacion	Nombre de la aplicación cliente (ejecutable).

Para el proceso servidor:

ruta	Especifica la ruta de directorios en donde serán creados los archivos del servidor
version	Es el número de versión de la política.
num_puerto	Número de puerto que se usará.
nombre_aplicacion	Nombre de la aplicación servidor (ejecutable)
dir_ip	Dirección IP del cliente

El protocolo de red a utilizar es el TCP a través del puerto especificado en el parámetro *num_puerto*.

El directorio que contendrá los archivos y scripts generados se creará dentro de la ruta especificada en el parámetro ruta.

5.1.2 Sintaxis del lenguaje de especificación definido en notación Backus-Naur form (BNF)

El Backus-Naur form (BNF) es una notación usada para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

A continuación se muestran la gramática del lenguaje de alto nivel, en BNF.

```
<startrule> ::= {<proceso>}

<proceso> ::= "process" <identificador> "{" {<asignacion>} "}"

<identificador> := {<letra> | <digit> |"_" | "-" |"."}

<asignacion> ::= <asigna_ruta> | <asigna_ver> | <asigna_npto> |
                 <asigna_app> | <asigna_IP>

<asigna_ruta> ::= "ruta" "=" <identificador>

<asigna_ver> ::= "version" "=" <digit>[{"." {<digit>}}]

<asigna_npto> ::= "num_puerto" "=" {<digit>}

<asigna_app> ::= "nombre_aplicacion" "=" {<letra> | <digito> |"_" | "-" |"."}

<asigna_IP> ::= "dir_ip" "=" <digito><digito><digito>"."
                <digito><digito><digito>"."<digito><digito><digito>"."
                <digito><digito><digito>

<letra> ::= "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"
            |"n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"|
            "A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"
            |"N"|"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"

<digito> ::= "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"0"
```

5.1.3 Compilador de la especificación de alto nivel

Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente, y lo traduce a otro programa equivalente en otro lenguaje, el lenguaje objeto [Aho90]. Un compilador es un programa que permite traducir el código fuente de un programa en

lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje de máquina u otro lenguaje de más bajo nivel que el del código fuente), de esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a como piensa un ser humano, algo que le resulte más natural y menos abstracto, para luego compilarlo a un programa más manejable por la computadora. Este proceso de traducción se conoce como **compilación**.

La construcción de un compilador involucra la división del proceso en una serie de fases que variará con su complejidad. Generalmente estas fases se agrupan en dos tareas: el análisis del programa fuente y la síntesis del programa objeto.

Análisis: Se trata de la comprobación de la correctitud del programa fuente, e incluye las fases correspondientes al análisis léxico, consiste en la descomposición del programa fuente en componentes léxicos o tokens; el análisis sintáctico, también conocido como parser, es la agrupación de los tokens en frases gramaticales; y el análisis semántico, es la comprobación de la validez semántica de las sentencias aceptadas en la fase de análisis sintáctico.

Síntesis: Su objetivo es la generación de la salida expresada en el lenguaje objeto.

A continuación se muestra la implementación de la notación BNF en Perl, esta implementación se hizo para un parser recursivo descendente, y se utilizó un módulo especial de Perl denominado Parse::RecDescent, disponible en el sitio web de CPAN (http://search.cpan.org) este módulo de Perl es necesario para implementar cada regla de la notación BNF como una expresion regular en Perl [Wall00], [Orwant02]. El conjunto de expresiones regulares componen la grámatica del parser (La regla para la dirección IP se tomó de [Fialka05]).

```
startrule      : proceso(s)

proceso        : 'process' identificador '{' asignacion(s) '}'
                  {main::mi_funcion(@parametros)}

identificador  : /[\w\_\-\.]+/
                 {$parametros[0]=$item[1];}

asignacion     : asigna_ruta | asigna_ver | asigna_npto | asigna_app |
                 asigna_IP

asigna_ruta    : 'ruta' '=' /\/([\w\_\-\.]+\/)*/
                 {$parametros[1]=$item[3];}
```

```
asigna_ver     : 'version' '=' /\d+(\.\d+){0,2}/
                  {$parametros[2]=$item[3];}

asigna_npto    : 'num_puerto' '=' /\d+/
                 { if((int($item[3]) > 65535) || (int($item[3]) < 0))
                   { print "\nERROR:", $item[3]," es numero de puerto invalido\n";
                     exit(0);
                   }
                   else {$parametros[3]=$item[3];}
                 }

asigna_app     : 'nombre_aplicacion' '=' /[\w\_\-\.]+/
                 {$parametros[4]=$item[3];}

asigna_IP      : 'dir_ip' '='/((25[0-5]|2[0-4][\d]|[01]?[\d][\d]?)\.){3}
                 (25[0-5]|2[0-4][\d]|[01]?[\d][\d]?)/
                  {$parametros[5]=$item[3];}
```

Como parte del proceso de implementación de MAC, se crean algunas variables importantes que son:

carpeta	Nombre del directorio que contendrá los archivos del módulo de política
nombre_modulo	Nombre del módulo de política
nombre_puerto	Nombre del tipo (type) del puerto. valor entre 0 y 65535
nombre_tipo	Nombre del tipo del servidor
nombre_tipoexe	Nombre del tipo ejecutable del servidor
nombre_firewall	Nombre del archivo del script de IPtables.

5.1.4 Traducción a especificaciones de bajo nivel

En esta implementación, una especificación escrita en el lenguaje de alto nivel, debe ser traducido como ya se habia mencionado, a una serie de especificaciones de bajo nivel, a saber:

- Archivos de Contexto.
- Archivos de Cumplimiento de Tipo.
- Scripts de comandos SElinux para compilación de política, etiquetado de archivos y de puerto.

- Script de reglas de configuración (script) de firewall para el filtrado de paquetes del kernel de Linux IPtables.
- Archivo de ejecución de firewall.

Todos estos elementos son creados a partir de la definición de la política de seguridad en el lenguaje de alto nivel, la cual lleva a cabo la creación de diversos archivos que contienen las especificaciones necesarias para la creación de la política del cliente y del servidor, y el script de IPtables, por medio de la compilación y ejecución de archivos, comandos y reglas. (ver Figura 5.3).

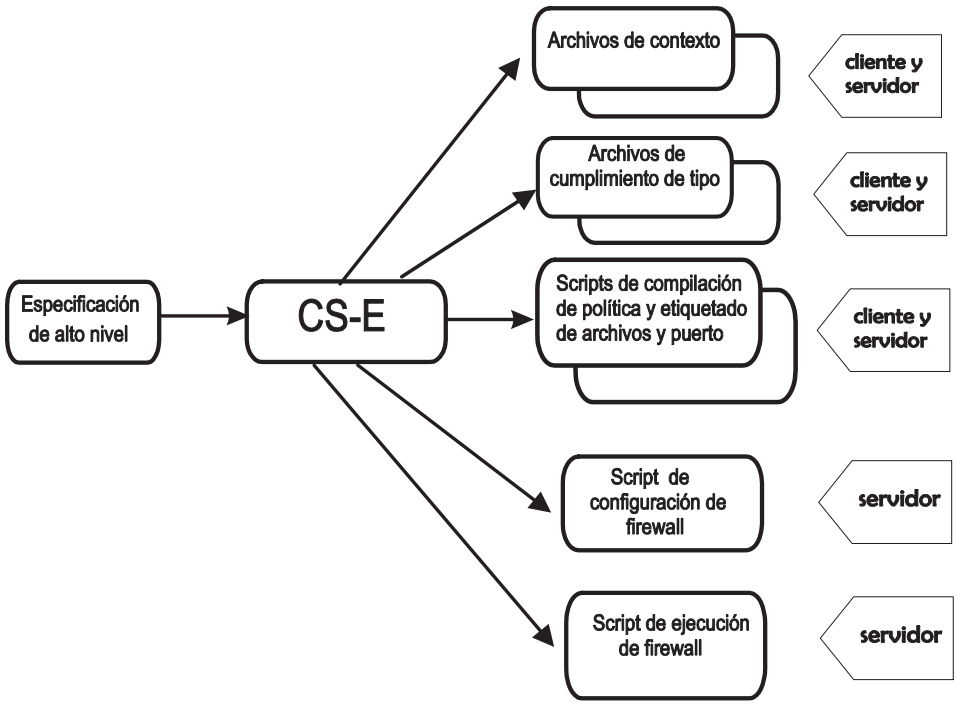

Figura 5.3: Operación de CS-E

En esta figura se ve como el proceso de implementación parte de la especificación de alto nivel, esta especificación va a alimentar a CS-E para la creación de los archivos de bajo nivel. Primero vienen los archivos para la política SELinux para el cliente y el servidor, los cuales son: el archivo de contexto y el archivo de cumplimiento de tipo, y se crean tanto

para el cliente como para el servidor. Estos archivos contienen declaraciones de los tipos a ser utilizados y las reglas que otorgan los permisos de los sujetos sobre los objetos, estas reglas hacer referencia a clases de objeto determinadas y a permisos determinados (no se van a dar todos los permisos de la clase del objeto) por eso estos deben ser cuidadosamente seleccionados para no otorgar más de los permisos que la aplicación requiere, ni menos de los necesarios (principio de privilegio mínimo). Aquí se hace uso de la granularidad en los permiso de acceso que el control de acceso TE en SELinux.

Se crean los scripts de compilación de políticas y etiquetado de puerto, estos scripts contienen diversos comandos para compilar e instalar la política del cliente y la política del servidor, reservar el puerto de comunicación y llevar a cabo el etiquetado de archivos.

Después viene la creación del script del firewall el cual contiene las reglas de filtrado de red acordes a lo especificado en la política de seguridad de alto nivel.

Y finalmente se crea el script del firewall, el cual contiene comandos para poner en marcha el filtrado de paquetes.

Con CS-E creamos cada unas de las sentencias correspondientes de cada uno de los archivos de bajo nivel que generamos, tanto los que pertenecen a la política SELinux y están expresados en el lenguaje de políticas de SELinux, los que pertenecen al script de IPtables expresados en reglas de filtrado de red y el conjunto de scripts de compilación y de ejecución que contiene comandos SELinux y Linux.

5.1.5 Creación de archivos de política SElinux

Se crean los scripts de compilación de políticas y etiquetado de puerto, estos scripts contienen diversos comandos para compilar e instalar la política del cliente y la política del servidor, reservar el puerto de comunicación y llevar a cabo el etiquetado de archivos.

Despúes viene la creación del script del firewall el cual contiene las reglas de filtrado de red acordes a lo especificado en la política de seguridad de alto nivel.

Y finalmente se crea el el script del firewall, el cual contiene comandos para poner en marcha el filtrado de paquetes.

Con CS-E creamos cada unas de las lineas de cada uno de los archivos de bajo nivel, tanto los que pertenecen a la política SElinux y estan expresados en el lenguaje de políticas de Selinux, los que pertenecen al script de IPtables expresados en reglas de filtrado de red y un conjunto de scripts de compilación y de ejecución.

Una política SELinux necesita basicamente de 2 archivos que son: el archivo de cumplimiento de tipo y el archivo de contexto o de etiquetado de política, ambos deben contener determinadas instrucciones en el lenguaje de politicas SELinux [technologyOpensourcesoftware08].Estos archivos deben estar presentes para poder compilar la política e instalarla. A continuación se explica con más detalle en que consisten estos archivos que CS-E tiene que crear como parte del proceso de implementación de los controles de acceso.

Archivo de cumplimiento de tipo

Este archivo incluye cosas como: la versión del módulo de política, la definición de los tipos a utilizar, sus atributos y el dominio al que pertenecen, los roles permitidos para esta política y finalmente vendrá una serie de declaraciones de las reglas en las cuales se especifica que es lo que los tipos declarados (sujetos), pueden hacer sobre las clases de objetos disponibles (objetos). Para ejemplificar esto se muestra continuación el contenido de un archivo de cumplimiento de tipo.
Ejemplo:

```
policy_module(servidor,1.0.0)
type  servidor_t;
type  servidor_exec_t;
type  servidor_port_t;

allow  servidor_t netif_type:netif {tcp_send};
allow  servidor_t netif_type:netif {tcp_recv};
```

La primera línea de este ejemplo define el nombre del archivo de contexto y el número de versión de política, despúes viene una defición de tipos y al final dos líneas en donde se especifican los permisos que primer tipo declarado tiene sobre las clases de objetos *netif_ type.*

Archivo de Contexto

El archivo de contexto es el encargado de proporcionar el etiquetado de los atributos de seguridad a los objetos, y para que pueda hacer esto, es necesario que se especifique la ruta de acceso de la aplicación a etiquetar.
Ejemplo:

```
/root/servidor/servidor_socket -- gen_context(system_u:object_r:servidor_exec_t)
```

5.1.6 Creación de scripts de instalación de política

Los scripts de instalación son un conjunto de instrucciones que van a ser ejecutadas directamente en la consola. Estas instrucciones incluyen comandos SElinux de compilación, instalación y verificación de politicas; así como un conjunto instrucciones de impresión de mensajes que informan de la aplicación de cada uno de los comandos.

5.1.7 Creación de script de configuración de firewall

Es en este script en donde se escriben las reglas para el filtrado de paquetes, es aquí en donde son útiles los valores de los parametros del número de puerto y la dirección IP del cliente, a continuación se muestra un ejemplo del contenido de este script .
Ejemplo:

```
/sbin/iptables -P INPUT    DROP
/sbin/iptables -P OUTPUT   DROP
/sbin/iptables -P FORWARD  DROP

/sbin/iptables -A INPUT  -i eth0  -s  192.168.2.129 -p tcp -d port 3550 -j ACCEPT
/sbin/iptables -A OUTPUT -o eth0  -d  192.168.2.129 -p tcp -s port 3550 -j ACCEPT
```

Lo que hacen las 3 primeras líneas de código del ejemplo es declarar una política por defecto DROP que solo acepta lo que esta explicítamente permitido, como es lo especificado en las 2 reglas posteriores de la cadena ACCEPT, que lo que hacen es aceptar la entrada

y la salida de paquetes a la direción IP 192.168.2.129 usando el puerto 3550 y el protocolo TCP. En una política por defecto DROP en las reglas de las cadenas ACCEPT, DROP o FORWARD.

5.1.8 Compilación e instalación de política

- **Compilación de la política.** Una vez creados los archivos para la política, estos deben ser compilados para crear la política, esto se hace por medio del comando *make* en el directorio en donde se encuentran los archivos, de la siguiente manera:

```
make -f /usr/share/selinux/devel/Makefile
```

- **Carga de la política.** Para cargar al kernel la nueva política creada y compilada, se usa el comando *semodule* con la opción *-i* y el nombre de módulo.

Ejemplo**:**

```
/usr/sbin/semodule   -i    myserver.pp
```

- **Etiquetado de puerto.** El puerto utilizado para la comunicación cliente-servidor es reservado y etiquetado con el comando *semanage* y la opcion *port*.

Ejemplo:

```
/usr/sbin/semanage port -a -t servidor_port_t -p tcp 7020
```

5.1.9 Etiquetado de archivos

Se aplican los contextos de seguridad a los directorios y/o archivos de la aplicación a la cual se la creó la política.
Ejemplo:

```
/sbin/restorecon -F -R -v root/Desktop/ServerGenFiles/servidor_socket2
```

5.1.10 Puesta en marcha del firewall

La puesta en marcha del firewall, se da al final de la ejecución del script del firewall. Primero se cambian los permisos del script para que pueda ser ejecutado, esto con el comando *chmod*.
Ejemplo:

```
chmod +x serverfirewall.sh
```

y después se ejecuta para ponerlo en marcha
Ejemplo:

```
sudo sh /root/Desktop/ServerGenFiles/mypolicy/serverfirewall.sh
```

5.2 Ejemplo Práctico de la utilización de CS-E

En este ejemplo práctico y real se hace la ejecución de CS-E, la cual muestra todos los pasos del proceso descrito anteriormente en este capítulo.

Inicialmente se muestra el contenido del archivo escrito en el lenguaje de especificación, que es de donde parte todo el proceso; despúes se muestra el contenido de cada uno de los archivos generados para el cliente y para el servidor: los de la política (archivos de contextoy archivos de cumplimiento de tipo), los script del firewall (configuración y ejecución) y los scripts de compilación.

Para el ejemplo se configuró una red en el sistema operativo fedora 9. Se designó un equipo servidor y dos clientes, con la siguiente configuración (Ver Figura **??**):

```
Red: 192.168.2.0/24
Mascara de red: 255.255.255.0
Direccion IP servidor: 192.168.2.128
Direccion IP cliente1: 192.168.2.129
Direccion IP cliente2: 192.168.2.130
```

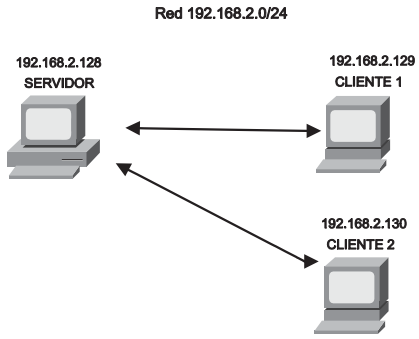

Figura 5.4: Red antes de aplicar CS-E

5.2.1 Definición de la política de seguridad para el cliente y el servidor

Código 1 defpolicy.txt

```
1:  process micliente
2:      {
3:          ruta =/root/Desktop/ClientGenFiles/
4:          version=1.0.0
5:          nombre_aplicacion = cliente_socket4
6:      }
7:
8:  process miservidor
9:      {
10:         ruta=/root/Desktop/ServerGenFiles/
11:         version=1.0.0
12:         num_puerto=7020
13:         nombre_aplicacion = servidor_socket4
14:         dir_ip=192.168.2.129
15:     }
```

El código 1 es la definición de la política de seguridad para el cliente y el servidor. Está

definición la hace el usuario en el lenguaje de alto nivel que se le proporciona para ello. Esta definición será la que será tomada por CS-E para la creación de los archivos de bajo nivel en SELinux e IPtables. A continuación se describe el contenido del código de la definición.
Las líneas 1 y 8 declaran los nombres de los procesos cliente y servidor respectivamente, que en el caso de este ejemplo son: *micliente* y *miservidor*. Estos nombres serán utilizados en distintas sentencias de los archivos de bajo nivel que se van a generar con CS-E.

En las líneas 3 y 10 se especifica la ruta en donde se encuentran los archivos ejecutables correspondientes al cliente y al servidor.

Las líneas 4 y 11 declaran el número de versión de la política, este valor va a ser usado por CS-E en la creación de los archivos de contexto del cliente y del servidor para asignarles su numero de version correspondiente.

Las líneas 5 y 13 especifican los nombres de los archivos ejecutables cliente y servidor respectivamente.

El proceso servidor contiene líneas adicionales para declarar parámetros que solo él utiliza, estos son: el número de puerto *num_puerto* en la línea 12, (es el que se utilizará en la comunicación con el cliente) y la dirección IP del cliente en la línea 14 (es la dirección del cliente seleccionado, para el caso del ejemplo es la 192.168.2.129.

El siguiente paso es la compilación de esta especificación con CS-E.

5.2.2 Archivos de bajo nivel generados por CS-E

A continuación ejecutamos CS-E para crear todos los archivos y scripts de bajo nivel necesarios para la implementación del control de acceso obligatorio. La forma de ejecutarlo es pasandole como parámetro de entrada el nombre del archivo que contiene la definición de seguridad para el cliente y el servidor, esto es, el archivo defpolicy.txt o código 1 de la siguiente manera:

```
[root@localhost ServerGenFiles]# ./CSEnhanced.pl defpolicy.txt
```

De esta manera generamos todos los archivos de bajo nivel necesarios para la implementación del control de acceso obligatorio, estos archivos son:

Archivos del servidor:

Código 2: miservidor.te

Código 3: miservidor.fc

Código 4: miservidorfirewall.sh

Código 5: installpolicy.pl (del servidor)

Archivos del cliente:

Código 6: micliente.te

Código 7: micliente.fc

Código 8: installpolicy.pl (del cliente)

Archivos generados para el servidor

Dentro de los archivos que generamos con CS-E para el servidor, están los que corresponden a la política en SELinux, que son el de cumplimiento de tipo y el de contexto (miservidor.te, miservidor.fc), ambos contienen sentencias escritas en el lenguaje de políticas de SELinux, y se muestran a continuación.

Código 2 miservidor.te

```
1:  policy_module(miservidor,1.0.0)
2:  ###################################
3:  # Declaraciones de tipos y roles
4:  ###################################
5:  # declaracion de los tipos
6:  type miservidor_t;
7:  type miservidor_exec_t;
8:  application_domain(miservidor_t, miservidor_exec_t)
9:  # asignacion del tipo creado al rol que lo puede usar
10: role system_r types miservidor_t;
11: # declaracion del tipo para el puerto a ser utilizado
12: type miservidor_port_t;
13: corenet_port(miservidor_port_t)
14: ###################################
15: # política local para servidor
16: ###################################
17: # la llamada a esta macro me permite reconocer el tipo netif_type
18: sysnet_dns_name_resolve(miservidor_t)
19: # permite crear el socket
20: allowmiservidor_t self:tcp_socket create_stream_socket_perms;
21: # permite hacer el send y el receive por la INTERFAZ de red
22: allow miservidor_t netif_type:netif { tcp_send };
23: allow miservidor_t netif_type:netif { tcp_recv };
24: # permite hacer el send y el receive por el NODO de red
25: allow miservidor_t node_type:node { tcp_send };
26: allow miservidor_t node_type:node { tcp_recv };
27: #permite hacer el send y el receive a traves del PUERTO especificado
28: allow miservidor_t miservidor_port_t:{ tcp_socket} {send_msg recv_msg };
29: #permite el BIND con el puerto especificado para la política
30: allow miservidor_t miservidor_port_t:tcp_socket name_bind;
31: allow miservidor_t node_type:{ tcp_socket} node_bind;
```

El código anterior (código2) generado con CS-E, corresponde al archivo de cumplimiento de tipo del servidor, en él aparecen declaraciones de tipos, rol al que se asignan y los permisos otorgados, a continuación se explica esto a detalle.

en línea 1, se asigna nombre a la política y su versión, estos datos se toman de los proporcionados en el código 1.

En las líneas 6 y 7 se hace la declaración de los tipos, en la línea 8 se hace uso de la función: *application_domain* para la transición de dominios de los tipos declarados.

En la línea 10 se asigna al rol *system_r* el tipo *miservidor_t*.

En la línea 12 y 13, se hace la declaración del tipo para el puerto de comunicación y por medio de la función *corenet_port* se le dan los atributos necesarios de un puerto.

En la siguiente parte del archivo se declaran por medio de las reglas allow, los permisos para los tipos. Los permisos son: el de la creación del socket (línea 20), permitir la recepción y el envío a través de la interfaz de red (línea 22 y 23), permitir la recepción y el envío a través del nodo (línea 25 y 26), permitir la recepción y el envío a través del puerto de red especificado (línea 28), y permitir hacer el bind para el puerto y nodo especificados (línea 30 y 31).

Código 3 miservidor.fc

```
1:  /root/Desktop/ServerGenFiles/servidor_socket4 - - gen_context(system_u:object_r:miservidor_exec_t,s0)
```

El código 3 es el archivo de contexto del servidor, es por medio de este archivo que se le proveé el contexto de seguridad a la aplicación servidor (usuario, rol, tipo y nivel de seguridad). En la única línea que contiene se especifica la ruta en donde se encuentra el archivo ejecutable del servidor, y se le asigna su contexto de seguridad por medio de la función *gen_context* para que de esa forma se indica quien puede hacer uso de él.

Código 4 miservidorfirewall.sh

```
1:  #!/bin/sh
2:  echo "Ejecutando script iptables del servidor"
3:
4:  #se hace FLUSH de las reglas
5:  /sbin/iptables -F
6:  /sbin/iptables -X
7:  /sbin/iptables -Z
8:
9:  # política por defecto que deniega todo (DROP)
10: /sbin/iptables -P INPUT DROP
11: /sbin/iptables -P OUTPUT DROP
12: /sbin/iptables -P FORWARD DROP
13:
14: #FILTRADO
15: #abrimos el puerto del servidor al que se va a conectar el cliente
16: # se establece la comunicacion bidireccional
17: /sbin/iptables -A INPUT -i eth0 -s 192.168.2.129 -p tcp –dport 7020 -j ACCEPT
18: /sbin/iptables -A OUTPUT -o eth0 -d 192.168.2.129 -p tcp –sport 7020 -j ACCEPT
19: echo "para verificar la configuracion aplicada ejecuta: /sbin/iptables -L -n "
```

El código 4, es un script que generamos con CS-E para la configuración de IPtables, éste contiene las reglas de filtrado necesarias para el tráfico entre el cliente y el servidor. Lo primero que se hace en este script es una limpieza de la tabla para eliminar cualquier configuración anterior (líneas 5,6 y 7).

En las líneas 10,11 y 12 se configura una política por defecto DROP.

En la línea 17, la regla especifica que se permite el tráfico entrante hacia la interfaz eth0, desde la dirección IP 192.168.2.129, que es el cliente definido, a través del puerto 7020 usando

el protocolo tcp.

De forma similar en la línea 18 se especifica que se permite el tráfico de salida por la interfaz eth0 hacia la dirección del mismo cliente, usando el mismo puerto.

A continuación se muestra el script de instalación del servidor.

Código 5 installpolicy.pl

```
1:  #!/usr/bin/perl
2:  print "—–>Compilando e instalando el modulo de política..... ";
3:  system("cd /root/Desktop/ServerGenFiles/miservidor; make -f /usr/share/selinux/devel/Makefile;
    /usr/sbin/semodule -i miservidor.pp");
4:
5:  print "—–>Listando el modulo de política";
6:  system(" /usr/sbin/semodule -l | grep miservidor");
7:  print " —–>Etiquetado de archivos..... ";
8:  system ("/sbin/restorecon -F -R -v /root/Desktop/ServerGenFiles/servidor_socket4;
9:  ls -Z| grep servidor_socket4");
10:
11: print " compilacion de política OK ";
12: print " —–> Asociando el puerto con su tipo..... ";
13: system ("/usr/sbin/semanage port -a -t miservidor_port_t -p tcp 7020");
14: print " etiquetado del puerto OK ";
15: print "—–>Listando el puerto asociado de la política";
16: print " nombre puerto protocolo numero de puerto ";
17: system ("/usr/sbin/semanage port -l | grep miservidor_port_t");
18: print "puerto listado OK ";
19: print "—–> Cambiando permisos al script de IPTABLES..... ";
20: system("cd /root/Desktop/ServerGenFiles/miservidor;pwd;chmod +x miservidorfirewall.sh;
21: ls -Z | grep miservidorfirewall.sh ");
22: print " OK. Permisos de script de IPTABLES cambiados ";
23: print "—–> Ejecutando el script de IPTABLES..... ";
24: system("sudo sh /root/Desktop/ServerGenFiles/miservidor/miservidorfirewall.sh");
```

El código 5, es un script que se genera para hacer la compilación e instalación de la

política del servidor. En la línea 3 se hace la compilación (por medio de la herramienta *make [Mecklenburg04]*) y la instalación de la nueva política (usando los comandos de SELinux). En la línea 8 se lleva a cabo el etiquetado de la aplicación para otorgarle su contexto de seguridad y en la línea 13 se asocia el puerto seleccionado con su tipo declarado en el código 2 . En la línea 20 se cambian los permisos del script y se pone a funcionar.

Archivos generados para el cliente

Código 6 micliente.te

```
1:  policy_module(micliente,1.0.0)
2:  ##################################
3:  # Declaraciones de tipos y roles
4:  ##################################
5:  #declaracion de los tipos
6:  type micliente_t;
7:  type micliente_exec_t;
8:  application_domain(micliente_t, micliente_exec_t)
9:  #declaracion de los roles
10: role system_r types micliente_t;
11: ##################################
12: # política local
13: ##################################
14: # la llamada a esta macro me permite reconocer el tipo netif_type
15: sysnet_dns_name_resolve(micliente_t)
16: #permite crear el socket
17: allow micliente_t self:tcp_socket create_stream_socket_perms;
18: # permite hacer el send y el receive por la INTERFAZ de red
19: allow micliente_t netif_type:netif  tcp_send ;
20: allow micliente_t netif_type:netif  tcp_recv ;
21: # permite hacer el send y el receive por el NODO de red
22: allow micliente_t node_type:node  tcp_send ;
23: allow micliente_t node_type:node  tcp_recv ;
```

El código 6 corresponde al archivo de cumplimiento de tipo del cliente, en él aparecen declaraciones de tipos, rol al que se asignan y los permisos otorgados, a continuación se

explica esto a detalle.

Así en la línea 1, al igual que en el archivo de contexto del servidor se asigna nombre a la política y su versión, estos datos se toman de los proporcionados en el código 1.
En las líneas 6 y 7 viene la declaración de los tipos a ser utilizados, y en la 8 se usa la función: *application_ domain* para la transición de dominios.
En la línea 10 se declara el rol que va a se utilizado y el tipo con que se va a usar.
En la línea 17 , la regla allow permite la creación del socket ; En la línea 19 y 20 se permite la recepción y el envío a través de la interfaz de red; y en la línea 22 y 23 se permite la recepción y el envío a través del nodo.

Código 7 micliente.fc

```
1:    /root/Desktop/ClientGenFiles/cliente_socket4 – gen_context(system_u:object_r:micliente_exec_t,s0)
```

El código7 es el archivo de contexto del cliente generado por CS-E, al igual que en el caso del servidor, es por medio de este archivo que se le asigna el contexto de seguridad a la aplicación cliente. La sentencia que allí aparece especifica la ruta en donde se encuentra el archivo ejecutable del cliente, a la cual se le asigna su contexto de seguridad por medio de la función *gen_ context*

Código 8 installpolicy.pl

```
1:  #!/usr/bin/perl
2:  print "—–>Compilando e instalando el modulo de política.....";
3:  system("cd /root/Desktop/ClientGenFiles/micliente; make -f /usr/share/selinux/devel/Makefile;
    /usr/sbin/semodule -i micliente.pp");
4:  print "—–>Listando el modulo de política";
5:  system(" /usr/sbin/semodule -l | grep micliente");
6:  print " —–>Etiquetado de archivos..... ";
7:  system ("/sbin/restorecon -F -R -v /root/Desktop/ClientGenFiles/cliente_socket4; ls -Z| grep
    cliente_socket4");
8:  print " compilacion de política OK ";
9:  print " »»política compilada e instalada ";
```

El código 8 es el script para la compilación del cliente, de forma similar al del servidor, contiene las líneas necesarias para la compilación e instalación de la política creada.
En la línea 3 se hace la compilación y la instalación de la nueva política del cliente.
En la línea 7 se lleva a cabo el etiquetado de la aplicación cliente.

5.2.3 Salidas por consola

En las siguientes líneas se muestran las salidas por consola al ejecutar CS-E con el archivo fuente.

Ejecución de CS-Enhanced

```
[root@localhost ServerGenFiles]# ./CSEnhanced.pl defpolicy.txt
estos son todos los valores de las variables dentro de la funcion
-----------------------------------------------------------------
$mypathname: /root/Desktop/ClientGenFiles/
$myfoldername: micliente
$mymodulename: micliente
$myversion: 1.0.0
```

```
$myportname:
$myportnum:
$mytypename: micliente_t
$myexetypename: micliente_exec_t
$myappname: cliente_socket4
$myfirewallname:
$diripcliente:
 -----> Creando archivo .te...
 OK
 ----- creando archivo .fc ...
 OK
 ----- creando script de compilacion ...
estos son todos los valores de las variables dentro de la funcion
----------------------------------------------------------------
$mypathname: /root/Desktop/ServerGenFiles/
$myfoldername: miservidor
$mymodulename: miservidor
$myversion: 1.0.0
$myportname: miservidor_port_t
$myportnum: 7020
$mytypename: miservidor_t
$myexetypename: miservidor_exec_t
$myappname: servidor_socket4
$myfirewallname: miservidorfirewall.sh
$diripcliente: 192.168.2.129
 -----> Creando archivo .te...
 OK
 ----- creando archivo .fc ...
 OK
 -----> Creando script para IPTABLES.....
 OK
 Ruta actual:
 ----- creando script de compilacion ...
```

Installpolicy del servidor

Se ejecuta el script de instalación del servidor

En las siguientes líneas se muestra la salida por consola que se generan al ejecutar el script de compilacion del servidor (installpolicy.pl).

```
[root@localhost miservidor]# ./installpolicy.pl
```

```
----->Compilando e instalando el modulo de politica.....
Compiling targeted miservidor module
/usr/bin/checkmodule: loading policy configuration from
tmp/miservidor.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 8) to
tmp/miservidor.mod
Creating targeted miservidor.pp policy package
rm tmp/miservidor.mod.fc tmp/miservidor.mod

----->Listando el modulo de politica
      miservidor 1.0.0

----->Etiquetado de archivos.....
/sbin/restorecon reset /root/Desktop/ServerGenFiles/servidor_socket4 context
unconfined_u:object_r:admin_home_t:s0->system_u:object_r:miservidor_exec_t:s0

 compilacion de politica OK

-----> Asociando el puerto con su tipo.....
 etiquetado del puerto OK

 ----->Listando el puerto asociado de la politica
nombre puerto       protocolo    numero de puerto
miservidor_port_t      tcp            7020

puerto listado OK

 -----> Cambiando permisos al script de IPTABLES.....
       /root/Desktop/ServerGenFiles/miservidor OK.

 Permisos de script de IPTABLES cambiados
-----> Ejecutando el script de IPTABLES.....
       Ejecutando script iptables del servidor

para verificar la configuracion aplicada ejecuta: /sbin/iptables -L -n

 Configuracion de firewall aplicada
Chain INPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 192.168.2.129 0.0.0.0/0 tcp dpt:7020
Chain FORWARD (policy DROP)
target prot opt source destination
```

```
Chain OUTPUT (policy DROP)
target prot opt source destination
ACCEPT tcp -- 0.0.0.0/0 192.168.2.129 tcp spt:7020
```

Se verifica que se haya creado el módulo del servidor.

```
[root@localhost ServerGenFiles]# semodule -l |grep miservidor
miservidor 1.0.0
```

Se verifica que se haya etiquetado el puerto

```
[root@localhost ServerGenFiles]# semanage port -l | grep miservidor_port_t
miservidor_port_t tcp 7020
```

Installpolicy del cliente

Se ejecuta el script de instalación del cliente. En las siguientes líneas se muestra la salida por consola que se generan al ejecutar el script de compilacion para el cliente.

```
[root@localhost micliente]# ./installpolicy.pl

----->Compilando e instalando el modulo de politica.....
 Compiling targeted micliente module
/usr/bin/checkmodule: loading policy configuration from tmp/micliente.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 8)
to tmp/micliente.mod
Creating targeted micliente.pp policy package
rm tmp/micliente.mod.fc tmp/micliente.mod
/etc/selinux/targeted/contexts/files/file_contexts:

----->Listando el modulo de politica
micliente 1.0.0

 ----->Etiquetado de archivos.....
 /etc/selinux/targeted/contexts/files/file_contexts:
 /root/Desktop/ClientGenFiles/cliente_socket4
(system_u:object_r:micliente_exec_t:s0 and system_u:object_r:cliente_exec_t:s0).
/sbin/restorecon reset /root/Desktop/ClientGenFiles/cliente_socket4
context system_u:object_r:cliente_exec_t:s0->system_u:object_r:micliente_exec_t:s0
```

```
compilacion de politica OK

>>>>Politica compilada e instalada
```

Se verifican los archivos creados en el directorio del cliente

```
 [root@localhost micliente]# ls
installpolicy.pl
micliente.fc
micliente.if
micliente.pp
micliente.te
tmp
```

Se verifica que se haya creado el módulo del cliente

```
[root@localhost ~]# semodule -l |grep micliente
micliente 1.0.0
```

Con la ejecución de los scripts *installpolicy.pl* del cliente y del servidor, se lleva a cabo la compilación e instalación de politicas, los etiquetados de archivos y puerto y la configuración del firewall, (Ver Figura 5.5).

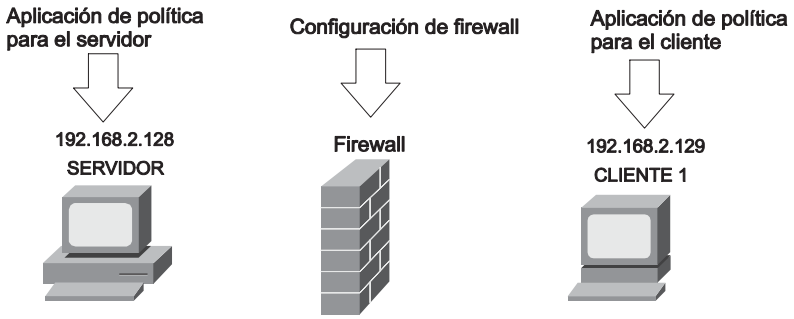

Figura 5.5: Ejecución de scripts

Una vez instaladas las politicas del cliente y el servidor y configurado el firewall, se ejecuta la aplicación cliente y la aplicación servidor.

Se ejecuta el servidor (192.168.2.130)

El cual queda en espera de las conexiones por parte de los clientes**.**

```
[root@localhost ServerGenFiles]# ./servidor_socket4 &
[1] 9701

[root@localhost ServerGenFiles]#
 ---->se ha hecho la correcta creacion del socket
---->se ha hecho el bind exitosamente
---->Servidor escuchando
---->se ha aceptado conexion
---->MENSAJE: archivo enviado correctamente.
--------------------------------------------------------------------------.
```

Se ejecuta el cliente 1 (192.168.2.129)

```
[root@localhost ClientGenFiles]# ./cliente_socket4 192.168.2.128 prueba.txt
--> Creacion del socket ha sido exitosa :-).
--> Conexion con el servidor ha sido exitosa :-)
-->Finaliza la conexion con el servidor
```

Se ejecuta el cliente 2 (192.168.2.130)

```
[root@localhost cliente_standart]# ./cliente_socket7020 192.168.2.128 prueba.txt
--> Creacion del socket ha sido exitosa :-).
ERROR: error el tratar de abrir un conexion.
```

La conexión es exitosa con el cliente para el cual se ha configurado el control de acceso (Cliente1, IP 192.168.2.129), en cambio fracasa cuando el otro cliente intenta hacer la conexión (Cliente2, IP 192.168.2.130) (Ver Figura 5.6).

Con este ejemplo práctico del uso de CS-E se ha mostrado de principio a fin, el funcionamiento de la herramienta, partiendo desde la definición en el lenguaje de especificación de alto nivel (que hace el usuario); la creación de los archivos de política para el cliente y el servidor; el script del firewall; los scripts de compilación del cliente y el servidor y la ejecución de dichos scripts. Las salidas por consola muestran el resultado de la ejecución.

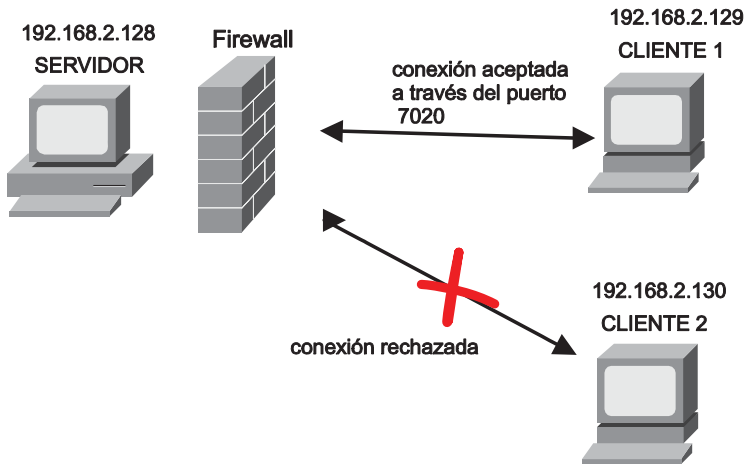

Figura 5.6: Esquema de la red con el control de acceso implementado.

5.3 Conclusiones

Este capítulo estuvo dedicado a explicar la estructura y el funcionamiento de CS-E. Se fué describiendo cada una de las partes que la constituyen y mostrando como es la relación funcional entre estas, para dar como resultado la construcción de la política de seguridad y la configuración del firewall para la aplicación Cliente-Servidor, cumpliendo de esta manera con el objetivo propuesto de la implementación de un control de acceso obligatorio en una arquitectura Cliente-Servidor.

CS-E resulta ser una herramienta útil, que le permite al usuario de manera sencilla, rápida, confiable y adecuada, generar una solución de seguridad para su aplicación Cliente-Servidor.

Una ventaja signficativa es la facilidad de uso que se tiene con CS-E, ya que para crear los archivos para la política y los scripts, no necesariamente se hace en los equipos destino, sino hasta el momento de la ejecución de los scripts de instalación correspondientes al cliente y al servidor.

CS-E constituye una herramienta útil para un usuario con poca o nula experiencia en el manejo de SELinux, ya que no es necesario que éste conozca los detalles importates y fuertes

que exige una configuración de este tipo.

CS-E le ahorra al usuario un gran tiempo de diseño, desarrollo e implementación, que seguramente le tomaría demasiado tiempo de prueba y error para aprender lo necesario para poder crear y aplicar correctamente una solución que le ayude a proteger a su aplicación Cliente-Servidor. Por mencionar las cosas que tendría que hacer el usuario si no utilizará CS-E, serian:

- Entender el funcionamiento de SELinux.
- Aprender el lenguaje de políticas de SELinux.
- Aprender a usar las clases, objetos e interfaces de SElinux.
- Conocer y manejar los comandos SELinux para la compilación e instalación de la política.
- Creación y ejecución del script de IPtables para el filtrado de la red.

Además, para cada nueva aplicación Cliente-Servidor tendría que editar (si ya los tiene, si no debe crearlos) todos los archivos de configuración para hacer cambios en los parámetros.

Con esta herramienta lo único que el usuario requiere hacer, es aprender el lenguaje de alto nivel desarrollado para crear el archivo fuente y a partir de allí utilizar a CS-E para que se encargue de crear los elementos necesarios para la protección de su aplicación Cliente-Servidor.

Para desarrollar este trabajo fue necesario conocer a fondo los distintos elementos de software que integran el proyecto, para poder diseñar, desarrollar, configurar, integrar y probar la herramienta creada, como por ejemplo:

- Manejo de sockets para la creación de la aplicación Cliente-Servidor.
- Conocer a fondo el manejo de SELinux (estructura, lenguaje, comandos, sistema de archivo, etc).
- Las reglas para los scripts de IPtables.

- La programación en lenguaje Perl.
- La sintaxis BNF.
- El módulo de Perl ParseRecDescent [Conway09] para el desarrollo del compilador.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones generales

El contenido de este trabajo expone la importancia de la seguridad en la información de toda organización, así como las amenazas a las que esta se ve expuesta.

Con los resultados obtenidos, se busca promover la creación de sistemas más seguros y robustos, que hagan uso del control de acceso obligatorio implementado a través de la herramienta generada, para las aplicaciones generadas bajo la arquitectura Cliente-Servidor sean capaces de soportar un gran número de amenazas a las cuales se ven expuestas durante su operación.

CS-E constituye un avance en cuanto a la utilización practica de SELinux, aprovechando las características de provisión de seguridad que ofrece, sumado el hecho de una protección a la red que es otro punto importante a tener en cuenta cuando se trata de asegurar un sistema computacional.

Se espera que con la facilidad de uso que CS-E ofrece, que no es más que introducir los valores necesarios para echar a andar todo el proceso, los administradores de red o cualquier usuario interesado en desarrollar aplicaciones seguras bajo la arquitectura Cliente-Servidor, adopte esta herramienta para lograr tener un alto grado de seguridad en sus desarrollos

Se desea que CS-E sirva como punto de partida para futuras mejoras y la creación de

nuevas aplicaciones de seguridad, que incorporen el uso del control de acceso obligatorio.

El resultado es una mejora sustancial en la seguridad de las aplicaciones Cliente-Servidor con un mínimo esfuerzo por parte del usuario, logrando de esta manera una mejora significativa en la implementación de una solución de seguridad contra daños y ataques.

Es importante porque se proveee un seguridad que es robusta pero se ofrece una implementacion muy sencilla al usuario, que no le demanda la necesidad de conocimientos técnicos ni gran inversión de tiempo para poder lograr mantener seguras a las aplicaciones Cliente-Servidor de las que hace uso.

6.2 Trabajos futuros

Como parte de los trabajos futuros se tiene pensado incluir herrmientas criptograficas como MD5 para un control de la integridad de los datos, ofrecer protección a la información que viaja a través del canal de comunicación entre el cliente y el servidor, y contar con una manera de hacer verificaciones de integridad de la información que se recibe para asegurarse que esta no ha sido alterada en el camino.

Otro punto sería proprcionar un manejo más amigable de la herramienta para facilitar aún más su utilización.

Además se ha considerado que se pueda definir más de una dirección IP, para permitir que la implementación del control de acceso obligatorio se aplique a una red completa.

Una opción más es integrar la configuración de IPtables como parte del arranque del sistema.

Bibliografía

[Aho90] Aho, A., Sethi, R., y Ullman, J. *Compiladores, principios, técnicas y herramientas*. Pearson Education, 1990. ISBN 9684443331.

[Badger95] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., y Haghighat, S. A. Practical domain and type enforcement for unix. *En Proceedings of the 1995 IEEE Symposium on Security and Privacy*, págs. 66–77. IEEE Computer Society, Glenwood, MD, USA, 1995. ISBN 1081-6011.

[Badger96] Badger, L., Sterne, D., y Sherman, D. A domain and type enforcement UNIX prototype. *The USENIX Association, Computing Systems*, 9(1):47 – 83, winter 1996.

[Bell73] Bell, L. L. Secure computer systems: mathematical foundations. *MITRE Technical Report*, 1(2547), March 1973.

[Bishop02] Bishop, M. *Computer Security: Art and science*. Addison Wesley, 2002. ISBN 0-201-44099-7.

[Cheswick94] Cheswick, A., S. Bellovin. *Firewalls and Internet Security:Repelling the Wily Hacker*. Addison Wesley Profes-

sional Computing Series. Second edition, 1994. ISBN 0-201-63466-X.

[Christiansen98] Christiansen, T. y Torkington, N. *Perl cookbook.Primera edición*. O'Reilly Media, 1998. ISBN 1-56592-243-3.

[Coker08] Coker. The guide to writing selinux policy. `http://www.linuxtopia.org/online-books/writing-SELinux-policy-guide/index.html`, 2008. Consulta: 2008.

[Conway09] Conway, D. Parse-recdescent. `http://search.cpan.org/~dconway/Parse-RecDescent`, 2009. Consulta:2009.

[DoD85] DoD. Departament of defense standart, trusted computer system evaluation criteria. departament of defense. December 1985.

[FedoraForum08] FedoraForum. Fedoraforum. `http://fedoraforum.org/`, 2008. Consulta:2008.

[Ferraiolo03] Ferraiolo, D., Kuhn, R., y Chandramouli, R. *Role-Based Access Control*. Computer Security Series. Artech House, 2003. ISBN 1-58053-370-1.

[Fialka05] Fialka, M. Match valid ip address. `http://vim.wikia.com/wiki/Match-valid-IP-address`, 2005. Consulta: 2009.

[Garcia09] Garcia, J. M. A specification language for information security policies. *En CIS'09: Proceedings of the international conference on Computational and information*

science 2009, págs. 437–440. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2009. ISBN 978-960-474-071-0.

[Lehtinen91] Lehtinen, R., Russell, D., y Gangemi Sr, G. *Computer Security Basics*. O'Reilly Media. Second edition, 1991. ISBN 978-0-937175-71-2.

[Loscocco98] Loscocco, P. The inevitability of failure: The flawed assumption of security in modern computing environments. *In Proceedings of the 21st National Information Systems Security Conference*, October 1998.

[MacAllister09] MacAllister, M., Walsh, D., Grift, D., Paris, E., Morris, J., y Radvan, S. *Fedora 11. Security enhanced linux user guide*. Linbrary, 2009.

[Mayer06] Mayer, F., MacMillan, K., y Caplan, D. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006. ISBN 0-131-96369-4.

[McCarty05] McCarty, B. *SELinux: NSA's Open Source Security Enhanced Linux*. OtŕReilly. Media, Inc, 2005. ISBN 0-596-00716-7.

[Mecklenburg04] Mecklenburg, R. *Managing Projects with GNU Make*. O'Reilly Media, 2004. ISBN 978-0-596-10445-0.

[Negus08] Negus, C. *Fedora 9 and Red Hat Enterprise Linux bible*. Wiley Publishing, Inc, 2008. ISBN 978-0-470-37362-0.

[Netfilter09] Netfilter, N., Firewalling y packet mangling for linux. The netfilter.org iptables project. `http://www.netfilter.`

org/projects/iptables/index.html, 2009. Consulta:2009.

[NIST07] NIST. National institute of standart and technology. http://csrc.nist.gov/, 2007. Consulta: 2007.

[NSA07] NSA. National security agency. http://www.nsa.gov/, 2007. Consulta: 2007.

[NSA09] NSA. Security enhanced linux. http://www.nsa.gov/research/selinux/index.shtml, 2009. Consulta: 2007.

[Orwant02] Orwant, J. *Computer Science and Perl Programming.* O'Reilly Media, 2002. ISBN 978-0-596-00310-4.

[Perl09] Perl. Perl.com the source for perl. http://www.perl.com/, 2009. Consulta: 2009.

[Petersen08] Petersen, R. *Fedora 9, Linux administration and security.* surfing turtle press, 2008. ISBN 0-9817778-1-8.

[Purdy04] Purdy, G. *Linux IPtables Pocket Reference.* O'Reilly Media, 2004. ISBN 0-596-00559-5.

[Randal03] Randal, A., Sugalski, D., y Tötsch, L. *Perl 6 Essentials.* O'Reilly Media, 2003. ISBN 0-596-00499-0.

[Rash07] Rash, M. *Linux Firewalls. attack, detection and response with IPtables, psand and fwsnort.* Linux firewalls, 2007. ISBN 978-1-59327-141-1.

[Red Hat07] Red Hat, I. Proyecto fedora. http://fedoraproject.org/es/get-fedora, 2007. Consulta: 2007.

[Sandhu94] Sandhu, R. y Samarati, P. Access controls: principles and practice. *IEEE communications magazine*, págs. 40–48, september 1994.

[Saunders01] Saunders, V., Hitchens. Role-based access control and the access control matrix. *ACM SIGOPS Operating Systems Review*, 35(4):6–20, October 2001.

[Schwartz97] Schwartz, R., Christiansen, T., y Wall, L. *Learning Perl*. O'Reilly Media. Second edition, 1997. ISBN 1-56592-284-0.

[SELinux07] SELinux. The selinux project userspace development. `http://userspace.selinuxproject.org/`, 2007. Consulta: 2007.

[Siever98] Siever, E., Spainhour, S., y Patwardhan, N. *Perl in a nutshell*. O'Reilly Media, 1998. ISBN 1-56592-286-7.

[Smalley01] Smalley, S. y Fraser, T. A security policy configuration for the security-enhanced linux. *NAI Labs*, February 2001.

[Smalley05] Smalley. Configuring the selinux policy. *NAI Labs Report*, 1(02-007), February 2005.

[Srinivasan97] Srinivasan, S. *Advanced Perl programming*. O'Reilly Media, 1997. ISBN 1-56592-220-4.

[Strand08] Strand, L. Rhel5 selinux: A benchmark. `http://blog.gnist.org/`, 2008. Consulta: 2008.

[Tanenbaum03] Tanenbaum, A. *Sistemas operativos modernos*. Pearson education. Segunda edición, 2003. ISBN 970-26-0315-3.

[technologyOpensourcesoftware08] technology Open source software, T. Getting started with reference policy. `http://oss.tresys.com/projects/refpolicy/wiki/GettingStarted`, 2008. Consulta: 2008.

[Tresys07] Tresys. Tresys technology. `http://www.tresys.com/selinux`, 2007. Consulta: 2007.

[Wall96] Wall, L., Christiansen, T., y Schwartz, R. *Programming perl*. O'Reilly Media. Second edition, 1996. ISBN 1-56592-149-6.

[Wall00] Wall, L., Christiansen, T., y Orwant, J. *Programming perl*. O'Reilly Media. Third edition, 2000. ISBN 978-0-596-00027-1.

Apéndice A

Código

```
use strict;
use Parse::RecDescent;
use Data::Dumper;

# Habilitar los  warnings dentro del modulo Parse::RecDescent
$::RD_ERRORS = 1; # Se asegura de que el parser muere cuando se encuentra un error
$::RD_WARN   = 1; # Habilita los  warnings
$::RD_HINT   = 1; # Proporciona pistas para ayudar a solucionar problemas

if($#ARGV= = -1)
{
  print "Uso: ./CSEnhanced.pl spe.txt";
  print "donde: spe.txt es el archivo con la especificacion de la politica de seguridad para el
  exit;
}

my $n1=$ARG[0];

use vars qw(@valores);

# Inicializacion de variables
my $mypathname=' ';        # servidor y cliente
my $myfoldername=' ';      # servidor y cliente
my $mymodulename=' ';      # servidor y cliente
my $myversion=' ';         # servidor y cliente
my $myportname=' ';        # servidor
my $myportnum=0;         # servidor
```

```
my $mytypename=' ';        # servidor y cliente
my $myexetypename=' ';     # servidor y cliente
my $myappname=' ';         # servidor y cliente
my $myfirewallname=' ';    # servidor
my $diripcliente=' ';     # servidor

#-----------------------------------------------------------
# Funcion que recibe los valores verificados por el parser
#-----------------------------------------------------------
sub mi_funcion
{
   # se reciben los valores de los tokens en variables separadas
  my ($mynameprocess,$mypathname,$myversion,$myportnum,
      $myappname,$diripcliente)=@_;

  # se asignan valores a otras variables a partir de las proporcionadas
  $myfoldername=$mynameprocess;
  $mymodulename=$mynameprocess;
  $mytypename=$mynameprocess."_t";
  $myexetypename=$mynameprocess."_exec_t";

  if ($myportnum!= 0)
  {
   # valores de variables solo para el servidor
    $myportname=$mynameprocess."_port_t";
    $myfirewallname=$mynameprocess."firewall.sh";
  }


 # impresion de los valores de todas las variables
  print "estos son todos los valores de las variables dentro de la funcion \n";
  print "-----------------------------------------------------------------\n";
  print "\$mypathname: ",$mypathname,"\n";
  print "\$myfoldername: ",$myfoldername, "\n";
  print "\$mymodulename: ",$mymodulename,"\n";
  print "\$myversion: ",$myversion,"\n";
  print "\$myportname: ",$myportname,"\n";
  print "\$myportnum: ",$myportnum,"\n";
  print "\$mytypename: ",$mytypename,"\n";
  print "\$myexetypename: ",$myexetypename,"\n";
  print "\$myappname: ",$myappname,"\n";
  print "\$myfirewallname: ",$myfirewallname,"\n";
  print "\$diripcliente: ",$diripcliente,"\n";
```

```
#*****************************************************************************
# CREACION DE ARCHIVOS PARA EL MODULO DE POLITICA SELINUX(cliente/servidor)
#*****************************************************************************
# creacion de la carpeta contenedora y acceso a ella
system("cd $mypathname;mkdir $myfoldername");

#-----------------------------------------------
# CREACION DEL ARCHIVO el archivo .te
#-----------------------------------------------
print "\n -----> Creando archivo .te... \n";

# creacion del archivo .te
open( PA, ">$mypathname$myfoldername\/$mymodulename.te");

#encabezado del modulo
print PA "policy_module($mymodulename,$myversion) \n";

 print PA "########################################  \n";
print PA "# \n";
print PA "# Declaraciones de tipos y roles  \n";
print PA "# \n";
print PA "######################################## \n\n";
print PA "#declaracion de los tipos\n";
print PA "type $mytypename;\n";
print PA "type $myexetypename; \n";
print PA "application_domain($mytypename, $myexetypename)\n\n";
print PA "#declaracion de los roles \n";
print PA "role system_r types $mytypename; \n\n";

# solo el servidor declara tipo para el puerto
if ($myportnum!= 0)
{
 print PA "#declaracion del puerto a ser utilizado\n";
 print PA "type $myportname;\n";
 print PA "corenet_port($myportname)\n\n";
}

print PA "########################################\n";
print PA "#\n";
print PA "#  politica local para servidor \n";
print PA "######################################## \n\n";
```

```
print PA "# la llamada a esta macro me permite reconocer el tipo netif_type\n";
print PA "sysnet_dns_name_resolve($mytypename)\n\n";

# declaracion de reglas allow
print PA "#permite crear el socket\n";
print PA "allow $mytypename self:tcp_socket create_stream_socket_perms;\n\n";

print PA "#  permite hacer el send y el receive por la INTERFAZ de red\n";
print PA "allow $mytypename netif_type:netif { tcp_send };\n";
print PA "allow $mytypename netif_type:netif { tcp_recv };\n\n";

print PA "#  permite hacer el send y el receive por el NODO de red\n";
print PA "allow $mytypename  node_type:node { tcp_send };\n";
print PA "allow $mytypename  node_type:node { tcp_recv };\n\n";

if ($myportnum!= 0)
{
    print PA "#permite hacer el send y el receive a traves del PUERTO specificado\n";
    print PA "allow  $mytypename $myportname:{ tcp_socket } { send_msg recv_msg };\n\n";

    print PA "# permite el BIND con el puerto especificado para la politica\n";
    print PA "allow $mytypename $myportname:tcp_socket name_bind;\n\n";

    print PA "allow $mytypename node_type:{ tcp_socket} node_bind;\n";
}

# termina la escritura al archivo de configuracion .te
# cerramos el archivo .te
close PA;

system(""); # quitar
print "\n OK \n";

#-------------------------------------------------
## CREACION DE ARCHIVO .fc
#-------------------------------------------------

print "\n ----- creando archivo .fc ...\n";

#creacion del archivo .fc
open( PB, ">$mypathname$myfoldername\/$mymodulename.fc");
```

```
# informacion para el etiquetado de archivos
print PB "$mypathname$myappname -- gen_context(system_u:object_r:$myexetypename,s0)\n";

# termina la escritura al archivo de configuracion .fc
# se cierrael archivo .fc
close PB;
print "\n OK \n";

#*********************************************************************
# CREACION DEL ARCHIVO (SCRIPT) DE IPTABLES PARA EL FIREWALL
#*********************************************************************

#-------------------------------------------
## CREACION DE ARCHIVO .sh (IPTABLES) (solo para el servidor)
#-------------------------------------------
#  La creacion de este es archivo es solo para el servidor
if ($myportnum!= 0)
{ # inicio del if

   print "\n -----> Creando script para IPTABLES..... \n";
   open( PC, ">$mypathname$myfoldername\/$myfirewallname");
   print PC "#!/bin/sh\n";
   print PC "echo \"Ejecutando script iptables del servidor\"\n\n";

   print PC "## se hace FLUSH de las reglas\n";
   print PC "/sbin/iptables -F\n";
   print PC "/sbin/iptables -X\n";
   print PC "/sbin/iptables -Z\n\n";

   print PC "# Politica por defecto que deniega todo (DROP)\n";
   print PC "/sbin/iptables -P INPUT DROP\n";
   print PC "/sbin/iptables -P OUTPUT DROP\n";
   print PC "/sbin/iptables -P FORWARD DROP\n\n";

   print PC "#FILTRADO\n";
   print PC "#abrimos el puerto del servidor al que se va a conectar el cliente\n";
   print PC "# se establece la comunicacion bidireccional\n";

   print PC "/sbin/iptables -A INPUT -i eth0 -s $diripcliente -p tcp
             --dport $myportnum -j    ACCEPT\n";
   print PC "/sbin/iptables -A OUTPUT -o eth0 -d $diripcliente -p tcp
             --sport $myportnum -j ACCEPT\n\n";
```

```
   print PC "echo \"para verificar la configuracion aplicada ejecuta:
             /sbin/iptables -L -n \"\n\n";

   # cerramos el archivo .fc
   close PC;
   print "\n OK\n";

   # me cambio al directorio donde esta ubicado el archivo
   print "\n Ruta actual:\n";
   #system("cd /root/Desktop/ServerGenFiles/mypolicy/;pwd;
             chmod +x serverfirewall.sh;
          ls -al| grep #serverfirewall.sh ");
} # fin del if($myportnum!= 0)



#****************************************************************************
## CREACION DEL SCRIPT PARA COMPILACION DE POLITICA (CLIENTE Y SERVIDOR)
#****************************************************************************
print "\n ----- creando script de compilacion ...\n";

#creacion del archivo de compilacion
open( PD, ">$mypathname$myfoldername\/installpolicy.pl");

# indico con que va a ser ejecutado
print PD "#!/usr/bin/perl\n";

#-------------------------------------------
## COMPILACION DE LA POLITICA
#-------------------------------------------
print PD "print \"----->Compilando e instalando  el modulo de politica..... \";\n";

print PD "system(\"cd $mypathname$myfoldername;
          make -f /usr/share/selinux/devel/Makefile;
  /usr/sbin/semodule -i $mymodulename.pp\");\n";

# Para verificar si ya cargado el modulo (nombre se modifica)
print PD "print \"----->Listando el modulo de politica\";\n";
print PD "system(\" /usr/sbin/semodule -l | grep $mymodulename\");\n";


#-------------------------------------------
## ETIQUETADO DE ARCHIVOS
```

```
#-------------------------------------------

print PD "print \" ----->Etiquetado de archivos..... \";\n";

 ls -al| grep servidor_socket2");

print PD "system (\"/sbin/restorecon -F -R -v $mypathname$myappname;
         ls -Z| grep $myappname\");\n" ;

# termina la escritura al archivo de compilacion
# se cierra el archivo
#close PD;
print PD "print \" compilacion de politica OK \";\n";


#-----------------------------------------------
## ETIQUETADO DEL PUERTO (solo para el servidor)
#-----------------------------------------------
if ($myportnum!= 0)
{
  # open( PE, ">$mypathname$myfoldername\/puertopol.pl");
   #  print PE "#!/bin/sh\n";
     print PD "\n print \" -----> Asociando el puerto con su tipo.....  \";\n";
     print PD "system (\"/usr/sbin/semanage port -a -t $myportname
                -p tcp $myportnum\");\n";
     print PD "print \" etiquetado del puerto OK \";\n";

    # Para verificar si ya cargo el puerto
    print PD "print \"----->Listando el puerto asociado de la politica\";\n";
    print PD "print \" nombre puerto   protocolo  numero de puerto \";\n";
    print PD "system (\"/usr/sbin/semanage port -l | grep $myportname\");\n";
    print PD "print \"puerto listado OK \";\n\n";
} #fin del if
#close PE;

# ------------------------------------------------------------------------------
# CAMBIO DE PERMISOS Y EJECUCION DE SCRIPT DE IPTABLES (solo para el servidor)
#-------------------------------------------------------------------------------
if ($myportnum!= 0)
{
     #open( PF, ">$mypathname$myfoldername\/installiptables.pl");
      #print PF "#!/bin/sh";
      print PD "print \"-----> Cambiando permisos al script de IPTABLES.\";\n";
```

```
     print PD "system(\"cd $mypathname$myfoldername;pwd;
               chmod +x $myfirewallname;ls -Z|
      grep    $myfirewallname.sh \");\n";
     print PD "print \" OK. Permisos de script de IPTABLES cambiados \";\n";

     #-------------------------------------------
     ## EJECUCION DE FIREWALL
     #-------------------------------------------
     print PD "print \"-----> Ejecutando el script de  IPTABLES.\";\n";
     print PD "system(\"sudo sh $mypathname$myfoldername\
                      /$myfirewallname\");\n";

     print PD "print \"  Configuracion de firewall aplicada \";\n";
     #revisar la configuracion actual del archivo
     print PD "system (\"/sbin/iptables -L -n \");\n";
} #fin del if

print PD "print \" >>>>Politica compilada e instalada \";";
close PD;
} # fin de mi_funcion


#****************************************************************************
## GRAMATICA
#****************************************************************************
my $grammar =q{
{my @parametros;}

    startrule : proceso(s)

    proceso : 'process' identificador '{' asignacion(s) '}'

    identificador : /[\w\_\-\.]+/
                   {$parametros[0]=$item[1];}

    asignacion : asigna_ruta | asigna_ver | asigna_npto | asigna_app
                    |asigna_IP
    asigna_ruta    :  'ruta' '=' /\/([\w\_\-\.]+\/)*/
                          {$parametros[1]=$item[3];}

    asigna_ver     : 'version' '=' /\d+(\.\d+){0,2}/
                    {$parametros[2]=$item[3];}
    asigna_npto    : 'num_puerto'  '=' /\d+/
```

```
                    {
                    if((int($item[3]) > 65535) ||
                           (int($item[3]) < 0))
                      {
                      print "\nERROR:", $item[3]," es numero de puerto invalido\n";
                      exit(0);
                      }
                     else
                     {
                     $parametros[3]=$item[3];}
                     }

  asigna_app     : 'nombre_aplicacion' '=' /[\w\_\-\.]+/
                   {$parametros[4]=$item[3];}

  asigna_IP      : 'dir_ip' '='/((25[0-5]|2[0-4][\d]|[01]?[\d][\d]?)\.)
                               {3}(25[0-5]|2[0-4][\d]|[01]?[\d][\d]?)/
                    {$parametros[5]=$item[3];}
};


# creacion de instancia del parser y asignacion de la gramatica
my $parser = Parse::RecDescent->new($grammar);
die("Gramatica incorrecta. \n") unless defined($parser);

# abriendo el archivo de la especificacion
open DATOS,"<$n1" or die;

#se le asigna al arreglo @datos el contenido del archivo
my @datos = <DATOS>;

# se sustituye el \n o \r del final de cada linea por un espacio en blanco
my $k;
foreach $k (@datos)
{
  $k=~s/[\n\r]/ /g;
}

my $datos = join("",@datos);

my $result = $parser->startrule($datos);
die("Texto de entrada incorrecto. \n") unless (defined ($result));
```

```
# Cerrando el archivo
close DATOS;

 ---------------------------------------------------------------------
```