Universidad Michoacana de San Nicolás de Hidalgo
Facultad de Ingeniería Eléctrica
Posgrado de Ingeniería Eléctrica

ARQUITECTURA PARA LA ADMINISTRACION DE POLITICAS DE SEGURIDAD SELINUX EN AMBIENTES DISTRIBUIDOS

TESIS

Que para obtener el grado de

DOCTOR EN CIENCIAS EN INGENIERIA ELECTRICA
OPCION SISTEMAS COMPUTACIONALES

presenta

Pedro Chávez Lugo

Juan J. Flores y Juan Manuel García García
Directores de Tesis

Septiembre 2009

Lista de Publicaciones

"User Profile Linear Correlation and Its Use on Intrusion Detection"
Pedro Chávez Lugo, Juan J. Flores, Juan Manuel García García
Mexican Conference on Informatics Security
México, D.F. 2006

"An Architecture for Security Policy Management on Homogeneous Networks"
Pedro Chávez Lugo, Juan J. Flores, Juan Manuel García García
Workshop of Computer Security on the Mexican International Conference on Artificial Inteligence
Aguascalientes, Aguascalientes, Mex. 2007

"Arquitectura de Seguridad para la Administración Sistematizada de Políticas SELinux en Ambientes Distribuidos"
Pedro Chávez Lugo, Juan J. Flores, Juan Manuel García García
Congreso Nacional de Ingeniería y Arquitectura IA08
Isnn: 9789689322405
Morelia, Mich., Mex. 2008

"Security Architecture for a Systematic Administration of SELINUX Policies in Distributed Environments"
Pedro Chávez Lugo, Juan J. Flores, Juan Manuel García García
Recent Advances in Data Networks, Communications, Computers
Isnn: 1790-5109
Bucharest, Romania 2008

"An Architecture for Systematic Administration of SELinux Policies in Distributed Environments"
Pedro Chávez Lugo, Juan J. Flores, Juan Manuel García García
International Journal of Computers and Communications
Isnn: 2074-1294
Reino Unido 2008

"A System for Distributed SELinux Policy Management"
Pedro Chávez Lugo, Juan Manuel García García, Juan J. Flores
3rd International Conference on Network and System Security NSS2009
Octubre 19-21 2009, Gold Coast Australia

Resumen

Esta tesis presenta una arquitectura para la administración de políticas SELinux aplicadas en un ambiente distribuido. La arquitectura propuesta consta de un servidor y un conjunto de clientes de política. El atributo localidad es utilizado para denotar a cualquier computadora del ambiente distribuido. Con este atributo se crean extensiones de regla que permiten especificar una relación localidad-roles que define el conjunto de roles que pueden ejercer los usuarios en una localidad l y una relación usuario-localidad-roles que define el conjunto roles que puede ejercer un usuario u en la localidad l.

El servidor de políticas tiene una política centralizada la cual, mediante una herramienta de segmentación, se traduce en varias políticas para las diferentes localidades del ambiente distribuido. La política es segmentada utilizando el atributo localidad y algunas extensiones a las reglas SELinux, propuestas en esta tesis. En cada proceso de segmentación se realiza el proceso de comparación de los valores *md5* correspondientes a las políticas segmentadas con los valores correspondientes de las políticas actuales de las localidades. Si existe diferencia entre el valor *md5* de la política segmentada y la política actual de una localidad, se realiza el proceso de actualización de política para esa localidad. Los procesos de comparación y actualización de política se realizan también cuando un cliente de política es encendido o restablecido. El sistema Kerberos es empleado para aprovechar las fases de autenticación y autorización utilizadas en el proceso de comparación y actualización de políticas entre el servidor y los clientes.

Abstract

This thesis presents an architecture for managing SELinux policies applied in a distributed environment. The proposed architecture consists of a server and a set of policy clients. The location attribute is used to denote any computer in the distributed environment. Based on this attribute, two new rule extensions are created to specify relationships between locations, users, and roles; a location-roles relationship defines the set of roles that users can play in a location, and a user-location-roles relationship defines the set of roles that an user can plays in a location.

The policy server contains a centralized policy, which using a segmentation tool is translated to policies for the different locations of the distributed environment. A policy is segmented using the location attribute and the SELinux rule extensions proposed in this thesis. After the segmentation process, an update process compares the md5 values for the segmented policies and the current location policies. If there is a difference between the md5 value for a segmented policy and the current location policy, then it is necessary to execute the policy update process for that location. The compare and update policy processes are executed when a computer is turned on or reset. The Kerberos systems is used for the authentication and authorization phases neccesary for the compare and update policy processes made between server and clients.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

S	Conjunto de Sujetos.
O	Conjunto de Objetos.
C	Conjunto de Clases.
$P(c)$	Conjunto de permisos para la clase c.
A	Conjunto de atributos.
T	Conjunto de Tipos.
R	Conjunto de Roles.
$\mathrm{D}(r)$	Conjunto de Roles dominados por el rol r.
U	Conjunto de Usuarios.
Cs	Conjunto todos los posibles contextos de seguridad.
Cv	Conjunto todos los contextos de seguridad válidos.
$M(s)$	Matriz de acceso TE.
$\gamma(s, o, p)$	Función para restricciones.
$\Delta(s, c)$	Función para el espacio de permisos autorizados.

Capítulo 1

Introducción

Las redes y los sistemas de cómputo constituyen una pieza importante para el desarrollo de la sociedad y son empleados en las áreas del comercio, salud, economía, educación, seguridad y comunicación. Esto trae como consecuencia una mayor inversión en la parte de seguridad informática debido a la exposición de información sensible perteneciente a un país, organización, o a una persona. La protección principal para la información sensible debe ser proporcionada por el sistema operativo, por lo cual es recomendable implementar en el kernel a los principales mecanismos básicos de seguridad.

Todo sistema operativo debe incluir mecanismos de seguridad que le permitan identificar usuarios, registrar eventos programados o anómalos y controlar el acceso a los recursos del sistema. Para la identificación de usuarios los sistemas operativos emplean un mecanismo de autenticación, que mediante un factor determina la validez de las identidades de usuario. Para el registro de eventos se emplea un mecanismo de auditoría, el cual a nivel del kernel registra las características importantes de los eventos de interés. Se emplea un mecanismo de control de acceso para controlar el acceso los recursos del sistema mediante la aplicación de modelos de control de acceso discrecional, obligatorio o ambos.

El control de acceso tiene la finalidad de regir la interacción entre usuarios y recursos. Al ser un intermediario tiene una gran responsabilidad para evitar la diseminación y/o alteración de información. Para resolver un gran número de problemas de seguridad los investigadores y desarrolladores han diseñado y desarrollado sistemas operativos que

fortalecen el control de acceso del kernel [Loscoco00]. Un ejemplo de este tipo sistema operativo es Linux que integra al control de acceso de SELinux (del inglés Security-Enhanced Linux), en el cual algunos modelos de control de acceso obligatorio son implementados a nivel del kernel; Cumplimiento de Tipo [typ09] [Badger95], Seguridad Multinivel [Ferraiolo92], una parte del modelo Basado en Roles [Ferraiolo92] y el modelo de Identidad de Usuario.

SELinux es un ejemplo del diseño y desarrollo del control de acceso, el cual no es una distribución de Linux sino más bien es el fortalecimiento del mecanismo de control de acceso del kernel. La interacción entre usuarios y recursos se define en una política de seguridad, la cual se especifica por un conjunto de reglas basadas en los modelos obligatorios que son empleados en el control de acceso. El proceso es la unidad fundamental del sistema operativo y para todo usuario en línea se le asocia un proceso. En términos de SELinux un proceso es asociado a un dominio y un recurso es asociado a un objeto. Una de las ventajas principales de SELinux es la alta granularidad para los permisos, ya que define 55 clases de objetos y 197 permisos de acceso.

Cada computadora con SELinux tiene asociado un conjunto de reglas (política) basadas en los modelos de control de acceso obligatorio y al utilizar varias computadoras con SELinux se presentan algunos problemas administrativos. Una política de SELinux puede estar formada por decenas de miles de reglas, esto trae como consecuencia grandes cambios en la operación y administración del sistema operativo Linux. Cuando las políticas sufren cambios, los administradores deben realizar las operaciones de modificación y actualización de políticas en todos los equipos involucrados. Para esto se deben emplear prácticas que permitan obtener *confidencialidad* e *integridad* en los procesos de modificación y actualización de políticas, ya que las directivas formales para las políticas SELinux no incluyen la especificación de ambientes distribuidos.

El sistema Kerberos puede ser utilizado para obtener seguridad en el proceso de actualización de políticas ya que en una red insegura dos o más computadoras pueden demostrar su identidad de una manera segura. Mediante el sistema Kerberos las diferentes computadoras en un ambiente distribuido pueden conocer y garantizar las identidades de otras computadoras.

1.1. Motivación

La **motivación** para llevar a cabo esta tesis surge de la inexistencia, en inglés o español de un trabajo que contenga la exposición y solución de algunos de los problemas que se presentan al emplear en los sistemas distribuidos a computadoras con SELinux/Linux. Dado que SELinux se encuentra en la fase de madurez es posible encontrar en la Internet una gran cantidad de información que no es clara para fines de aprendizaje. Existen los libros *SELinux NSA's Open Source Security Enhanced Linux* [McCarty04] y *SELinux by Example: Using Security Enhanced Linux* [Frank Mayer06], los cuales describen los aspectos básicos de SELinux desde el punto de vista operativo y administrativo, y no presentan la integración de SELinux en los ambientes distribuidos. Además de existir muy pocos trabajos de investigación que aplican a SELinux en ambientes distribuidos.

Por ello, en mi opinión es necesario contar con una referencia que describa claramente los aspectos básicos del control de acceso, para abordar de una manera sencilla la parte operativa y administrativa de SELinux. Tal referencia debe incorporar posibles soluciones a los problemas que se presentan en los ambientes distribuidos que aplican a SELinux. En la Tabla 1.1, se muestra un comparativo entre el presente trabajo nombrado *Arquitectura para la Administración de Políticas SELinux en Ambientes Distribuidos* (A^2PSAD) y *SELinux Policy Management and Distribution Prototype* (PDM) [pdm08]. La característica de mayor importancia corresponde a la administración de identidad/localidad, la cual es incorporada en el presente trabajo mediante el sistema Kerberos. Lo cual permite que los diferentes elementos de la arquitectura propuesta tengan asociada una identidad, además de poder conocer y garantizar al resto de las identidades. El tipo de política es otra característica importante a resaltar ya que una política modular es aplicada para encapsular o confinar a los procesos que corresponden a los servicios de red para que solo puedan acceder a los recursos necesarios para su operación. Con este tipo de política se evita un análisis del resto de las reglas que conforman la política. Por otro lado la política monolítica es empleada en los sistemas orientados a las organizaciones, ya que concentra al conjunto de reglas de la política, razón por la cual se verifica la dependencia entre reglas. Adicionalmente en este trabajo se define una regla extendida de SELinux, la cual permite especificar roles y

usuarios para cada identidad definida mediante el atributo localidad.

Tabla 1.1: Comparación entre AAPSAD y PDM

Característica	AAPSAD	PDM
Tipo de política	Monolítica	Modular
Política centralizada	√	√
Redundancia en servidor	×	×
Registro de auditoría	×	√
Administración de identidad	Kerberos	×
Reglas de localidad	√	×

En resumen, creo que este trabajo está justificado por que contiene aspectos distintos y novedosos frente a los libros y trabajos de investigación existentes sobre SELinux.

1.2. Hipótesis

Mediante una arquitectura que agrupe y segmente políticas se puede llegar a resolver el problema de la administración de políticas SELinux aplicadas en los ambientes distribuidos. Para lograrlo se analizará la estructura de las políticas de SELinux en busca de una alternativa de especificación que permita obtener una política centralizada, la cual contenga a todas las políticas del ambiente distribuido. La política centralizada deberá ser segmentada en políticas individuales, las cuales serán asignadas a cada computadora del ambiente distribuido. Se pretende implementar una arquitectura compuesta por un servidor de políticas y un conjunto de clientes de políticas; tales elementos serán especificados mediante la identidad basada en el atributo de localidad. El sistema Kerberos será integrado a esta solución para intentar obtener la identificación de las diversas identidades del ambiente distribuido, así como también garantizar la confidencialidad e integridad de las políticas en los procesos de asignación y actualización.

1.3. Planteamiento del Problema

Una política SELinux esta formada por un gran número de reglas y al emplear computadoras con Linux/SELinux se obtienen cambios significativos en la operación y administración del sistema operativo. En un ambiente distribuido, cuando las políticas sufren cambios los administradores deben realizar manualmente los procesos de modificación y actualización de políticas en todos los equipos involucrados. Un administrador puede por alguna razón omitir modificar y actualizar uno o más políticas. Además de ser necesario que los administradores hagan uso de técnicas que provean integridad y confidencialdad en los procesos de modificación y actualización.

1.4. Objetivos de la Tesis

El presente trabajo intenta solucionar los diferentes problemas que se presenten en los ambientes distribuidos que emplean sistemas SELinux.

1.4.1. Objetivo general

Ofrecer una arquitectura que provea una base sólida para la administración sencilla y confiable de políticas SELinux en los ambientes distribuidos.

1.4.2. Objetivos particulares

- Ofrecer un marco de referencia del control de acceso para los tipos y modelos de control de acceso de mayor importancia.
- Citar algunas deficiencias de las diferentes especificaciones empleadas en el control de acceso tradicional que no consideran el mal uso de recursos.
- Ofrecer reglas extendidas de SELinux que permitan la especificación formal de los ambientes distribuidos.
- Desarrollar una herramienta para la segmentación y actualización de políticas con la finalidad de minimizar la inconsistencia entre las políticas del servidor y los clientes.

- Integrar al sistema Kerberos para obtener la confidencialidad, integridad y autenticidad en el manejo de políticas SELinux en ambientes distribuidos mediante las fases de Autenticación, Autorización y Servicio.

1.5. Descripción de Capítulos

En el presente Capítulo 1 se presento una introducción y se definió el problema central que se aborda en esta tesis citando los objetivos de esta misma. En el capítulo 2 se presenta la revisión del estado del arte citando y describiendo algunos trabajos de las diversas líneas de investigación del control de acceso. Previo a esto se da una introducción de los diferentes mecanismos de seguridad, haciendo énfasis del control de acceso para lo cual se definen los tipos de control de acceso y se describen los principales modelos del control de acceso. En el Capítulo 3 se da una introducción a los sistema SELinux describiendo su estructura, operación y las directivas empleadas en la construcción de políticas. Se cita un modelo formal para SELinux y se describen algunas de las deficiencias que se obtienen al aplicarlo a SELinux en ambientes distribuidos. El Capítulo 4 describe la arquitectura propuesta empleada en ambientes distribuidos para administrar a los sistemas SELinux de una manera sencilla y segura. Se describe al atributo de localidad y se presentan las extensiones de reglas necesarias para la segmentación de políticas. Se emplea al modelo formal presentado en el Capítulo 3 para incluirle la especificación de localidades. En el Capítulo 5 se presentan los resultados obtenidos del desarrollo e implementación del presente trabajo. Finalmente en el Capítulo 6 se presentan las conclusiones derivadas de esta investigación y se plantean los trabajos para futuras investigaciones.

Capítulo 2

Revisión del Estado del Arte

En este capítulo se presentan algunos trabajos de las diversas líneas de investigación del control de acceso. Previo a esto se da una introducción de los diferentes mecanismos de seguridad, haciendo énfasis del control de acceso. Por el cual se definen los tipos de control de acceso y se describen los principales modelos del control de acceso.

2.1. Mecanismos de Seguridad

A un alto nivel de abstracción, las personas emplean los sistemas de cómputo para desarrollar sus tareas o actividades diarias mediante una entidad activa del sistema operativo denominada *proceso*. Los sistemas de cómputo tienen asociados diferentes *recursos* (archivos, directorios, tuberías, sistemas de archivos, sockets, etc.), los cuales son empleados por los procesos. En términos del sistema operativo el proceso y recurso se llaman *sujeto* y *objeto*, respectivamente.

Un sistema operativo debe contar con mecanismos de seguridad que permitan identificar usuarios, limitar el acceso a los recursos, y registrar acciones desarrolladas por los usuarios. Tales mecanismos deben aplicarse en el momento requerido, y deben contar con una protección que garantice su uso. Algunos ejemplos de tales mecanismos son:

- Autenticación.

- Auditoría.

- Control de acceso.

La pregunta obligada en la prueba de estos mecanismos, es el cómo determinar si un sistema que los emplea es seguro. Para responder esta pregunta algunas organizaciones forman equipos de especialistas que tienen la finalidad de obtener acceso no autorizado a los recursos, ya que la principal ventaja de un atacante es conocer y aplicar técnicas que le permiten descubrir y aprovechar vulnerabilidades que no son conocidas por quienes lo diseñaron, o administran un sistema.

Bob Toxen [Toxen00] identifica varias prácticas de inseguridad que son comúnmente realizadas por los administradores de sistemas. A tales prácticas Toxen les denomina los "siete pecados mortales":

- Contraseñas débiles.

- Puertos de red abiertos.

- Versiones no actualizadas de software.

- Seguridad física pobre.

- CGIs (del Inglés *Common Gateway Interfce*) inseguros.

- Cuentas viejas e innecesarias.

- Demora en actualizaciones por parte del administrador.

Un sistema operativo debe ser administrado correctamente para minimizar el riesgo de las posibles amenazas que puedan afectar la confidencialidad, integridad y disponibilidad de los recursos. Las estructuras y los mecanismos del sistema operativo deben estar basados en un modelo formal de seguridad y para su implementación es conveniente seguir los criterios de diseño, implementación certificación y acreditación, establecidos por instituciones como el DoD con el "Libro Naranja"[dod85a], o como los citerios comunes de la ISO [iso08].

2.1.1. Mecanismo de Autenticación

Todo sistema operativo debe contar con la capacidad de identificación de usuarios y crear para cada usuario válido una sesión de operación que le permita interactuar con los recursos del sistema. El mecanismo de autenticación es la primera barrera técnica de seguridad y tiene como finalidad verificar la identidad de los usuarios. La autenticación puede estar basada en uno o más de los siguientes factores:

- Algo que se conoce.
- Algo que se tiene.
- Algo que se es ó Biométrico.

Factor algo que se conoce

El factor "algo que se conoce" se convierte en el secreto compartido entre usuario y sistema; tal secreto puede consistir de un número, palabra o frase. El sistema debe poseer al secreto compartido, aunque la experiencia ha demostrado que se corre un riesgo al tenerlo registrado en un medio de almacenamiento. Para ésto se emplean los algoritmos de resumen (md5 [Group92], sha [ofStandards93], etc.), los cuales generan una cadena de bits correspondiente a la huella ó resumen de la frase secreta. Este factor es tradicionalmente empleado debido a su fácil implementación y bajo costo. Al emplear este factor se deben eliminar los ataques de adivinanza, fijando un límite de intentos de autenticación, además de no informar la invalidez de usuarios.

Factor algo que se tiene

El factor "algo que se tiene" posee un bajo nivel de fiabilidad debido al empleo de partes físicas susceptibles a extravío o robo, por tales razones debe ser combinado con los factores "algo que se conoce" ó "biométrico". Las tarjetas magnéticas, generadores de números aleatorios son algunos ejemplos para el factor algo que se tiene.

Factor algo que se es ó Biométrico

El factor biométrico considera que los usuarios pueden reproducir sus características físicas

de una manera repetitiva y precisa. Para este factor existen en el mercado una amplia gama de lectores biométricos de diversos tipos y marcas. Se pueden encontrar sensores de huella dactilar, retina, iris, contorno de mano, etc., los cuales pueden relizar búsquedas de *uno a muchos* ó *uno a uno*.

El número de factores empleado por el mecanismo de autenticación se basa normalmente en la sensibilidad de la información contenida en el sistema de cómputo. Un mecanismo de autenticación robusto combina dos o más factores, lo cual implica un costo alto de implementación y mantenimiento. En general proteger información sensible implica una mayor inversión de recursos (tiempo y dinero) en la implementación de los mecanismos de seguridad, aunado a un posible aumento de complejidad para la operación y administración. Algunos trabajos sobre mecanismos biométricos, administradores de contraseña, y tarjetas magnéticas pueden encontrarse en [Lucas Ballard06, Sonia Chiasson06, Drimer07, Valentin Sgarciu06].

2.1.2. Mecanismo de Auditoría

El mecanismo de auditoría tiene la meta de registrar diversos eventos realizados por los usuarios y el propio sistema. Los diseñadores, programadores, y administradores necesitan analizar los registros de auditoría para intentar resolver problemas de seguridad. La identidad, acción, y el tiempo son los aspectos mínimos a registrar. Los registros de eventos deben contener la información necesaria para la:

- Reconstrucción cronológica de eventos.

- Detección de eventos no autorizados.

- Identificación de problemas de configuración.

Un sistema puede registar un gran número de actividades y para evitar la carencia de espacio en medios de almacenamiento los registros de auditoría deben limitarse, además de ser necesario una constante depuración. Más información detallada sobre los registros de

auditoría se puede encontrar en [Swanson96].

Aspecto legal

Los sistemas auditores pueden tener la capacidad de registrar eventos locales y/o remotos. Las organizaciones deben contar con el respaldo legal que les permita registrar, analizar y sancionar a los miembros que incurran en actividades no permitidas, intentado con ésto no afectar la estabilidad de la organización. En nuestro país existen algunos intentos de abordar a la legislación informática, de tal manera que el código penal federal en su artículo 211 de los incisos 1 al 7 [cpf09] abordan el acceso ilícito a sistemas y equipos de informática. Algunos estados por su parte han relizado esfuerzos para cubrir a la legislación informática, como ejemplo citamos al código penal del Distrito Federal [cpd07], código penal del estado de Sinaloa [cps06], y la Ley de protección de datos del estado de Colima [lpd03]. En nuestro país es necesario un mayor esfuerzo para regular los delitos informáticos y de manera global también es necesario integrar la reglamentación informática que involucre a todos los países del mundo, ya que las comunicaciones electrónicas han venido a modificar a la interacción entre las naciones y sus pobladores.

2.1.3. Mecanismo de Control de Acceso

Un sistema operativo cuenta con un conjunto de sujetos o usuarios y un conjunto de objetos o recursos. Esto implica que se debe regular la interacción entre los elementos de tales conjuntos, y para ello el mecanismo de control de acceso tiene la finalidad de limitar la interacción entre sujetos y objetos. La autorización es parte del control de acceso y su función es otorgar o negar el acceso a un objeto por parte de la acción de un sujeto. La Figura 2.1, muestra un sujeto s_1 que pretende tener acceso de lectura sobre el objeto o_1 y dependiendo de las reglas de la autorización el acceso puede ser otorgado o negado.

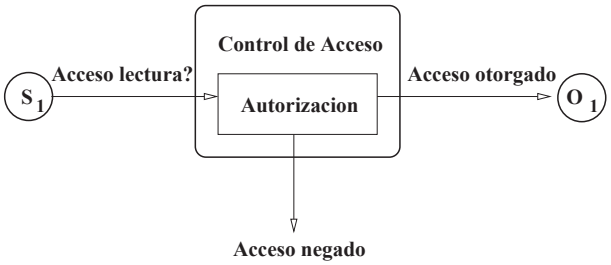

Figura 2.1: Mecanismo de control de acceso

Morrie Gasser [Gasser88] cita tres tareas fundamentales para el mecanismo de control de acceso:

- *Autorización.* Determina si un sujeto tiene el privilegio de tener acceso sobre un objeto.
- *Determinar los derechos de acceso.* Los derechos de acceso se obtienen de la combinación de los modos de acceso tales como lectura, escritura, ejecución, etc.
- *Cumplimiento de los derechos de acceso.*

El control de acceso establece que los sujetos y objetos deben tener asociado un conjunto de *atributos de seguridad* que garantice la validez de acceso. La determinación de acceso a un objeto se fundamenta en tales atributos y en una *política* de seguridad que consiste de un conjunto de reglas basadas en uno o más modelos de control de acceso. A continuación se describe al antecedente del control de acceso definido por el monitor de referencia.

2.1.4. Monitor de referencia

Un estudio de planificación tecnológica para la seguridad de cómputo [Anderson72], detalla los requerimientos de seguridad de los sistemas de cómputo de la Fuerza Aérea de los Estados Unidos. Con los sistemas compartidos se originaron problemas de seguridad y privacidad, tales como el manejo seguro de información clasificada, interconexión de sistemas de cómputo, penetración de usuarios internos y/o externos. Para resolver estos problemas de seguridad, el estudio propone que los sistemas se integren con los siguientes componentes:

- Un mecanismo adecuado de control de acceso.

- Un mecanismo de autorización.

- Ejecución controlada de programas de usuario.

El monitor de referencia intenta validar todas las referencias realizadas por los programas en ejecución, garantizando que la referencia sea del tipo correcto. Para ésto se propone el mecanismo de validación de referencia (RVM) como la implementación del monitor de referencia, bajo los siguientes requerimientos:

- Ser resistente a las alteraciones.

- Siempre debe invocarse.

- Suficientemente pequeño para ser sujeto a análisis y garantizar su validez.

El monitor de referencia constituye el origen del control de acceso, tal como se muestra en la Figura 2.2, en la cual el monitor de referencia se encarga de regular los accesos a programas, datos y dispositivos de E/S. Actualmente el control de acceso reemplaza al esquema del monitor de referencia.

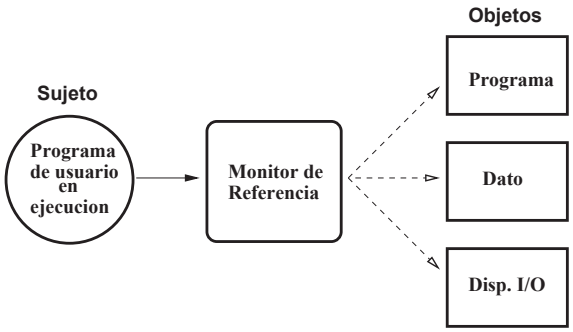

Figura 2.2: Monitor de Referencia

2.2. Tipos de Control de Acceso

El tipo de control de acceso describe la definición conceptual del control de acceso. El documento que contiene al criterio de evaluación para sistemas confiables de cómputo

[DOD85b] o mejor conocido como libro naranja, cita dos diferentes tipos de control de acceso:

- *Control de Acceso Discrecional (DAC).*
- *Control de acceso Obligatorio o no discrecional (MAC).*

Las siguientes definiciones de los tipos de control de acceso son tomadas del libro naranja [DOD85b].

Control de Acceso Discrecional
Es un medio para restringir el acceso a los objetos basado en la identidad de los sujetos y/o grupos para los cuales han sido otorgados. Este tipo de control es discrecional en el sentido de permitirle a un sujeto con ciertos permisos de acceso la capacidad de otorgar permisos a otros usuarios.

Las reglas del control de acceso discrecional permiten que los usuarios cambien o modifiquen los atributos de sus objetos.

Control de Acceso Obligatorio
Es un medio para restringir el acceso a los objetos basado en la sensibilidad de la información contenida en los objetos y la autorización formal de los sujetos para acceder a la información con una cierta sensibilidad que representa el grado de importancia.

Las reglas del control de acceso obligatorio no permiten que los usuarios cambien o modifiquen los atributos de sus objetos ya que las reglas son controladas por una organización.

2.3. Modelos de Control de Acceso

Un modelo de control de acceso describe la definición formal y concreta del control de acceso. Un modelo formal de control de acceso es un componente importante que ofrece

una base para el diseño y construcción de sistemas confiables. Es recomendable implementar en el espacio del kernel a los modelos de control de acceso y evitar usar el espacio de usuario en la solución de todos los problemas de seguridad. El modelo formal ayuda a demostrar la seguridad de una implementación, el cual debe ser implementado bajo técnicas eficientes de programación. Landwehr [Landwehr81] cita algunos modelos formales desarrollados desde 1970 a 1980. No es de nuestro conocimiento de algún documento que contenga el resumen de diversos trabajos del control de acceso a partir de 1980 a la fecha.

2.3.1. Modelo de Matriz de Acceso

Lampson [Lampson71] propuso el uso de la matriz de control de acceso; su propuesta está compuesta por cuatro conjuntos: Sujetos (S), Objetos (O), combinación de lectura-escritura-ejecución representada por un modo de acceso (R), y una matriz que representa cómo y cuáles objetos y sujetos pueden ser accedidos por un sujeto.

La Figura 2.3 muestra una matriz de acceso y las notaciones para sujetos, objetos y el modo de acceso. Un objeto O_b o sujeto S_c puede ser accedido bajo el modo de acceso r_j por el sujeto S_a si y sólo si A[S_a,$O_b \vee S_c$] = r_j, donde $r_j \in$ R.

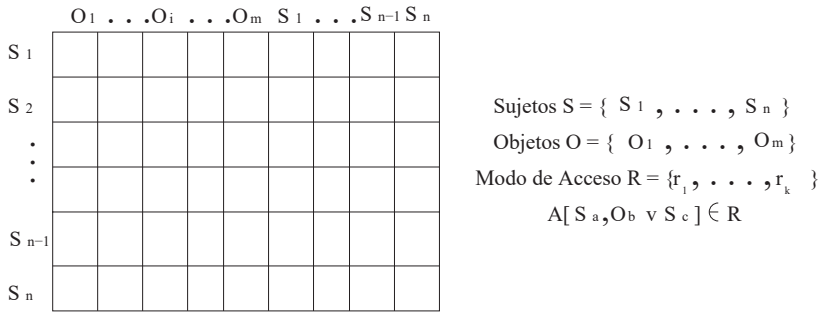

Figura 2.3: Matriz de Acceso

Actualmente, una variante del modelo de matriz se llama modelo basado en permisos del sistema de archivos y es usada en los sistemas operativos Unix y Linux desde los años setentas y ochentas, respectivamente [Grunbacher03]. Para esta variante, cada objeto tiene asociados tres conjuntos de permisos que definen el acceso del propietario, grupo del propietario, y los otros usuarios. Cada conjunto contiene los permisos lectura(r), escritu-

ra (w), y ejecución (x), los cuales son representados por nueve bits. En la Figura 2.4 se muestra los bits representativos de los conjuntos de lectura-escritura-ejecución aplicados al propietario, grupo y los otros.

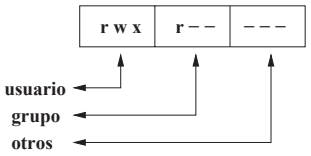

Figura 2.4: Bits del modo de acceso

El modelo basado en permisos del sistema de archivos sigue siendo utilizado debido a su simplicidad que ofrece una fácil implementación, administración y uso. Una desventaja de éste modelo es la falta de granularidad; aplicado al usuario, grupo y otros. Este modelo es del tipo de control de acceso discrecional, donde los sujetos pueden cambiar el atributo de sus objetos. Para algunas operaciones de los sujetos es necesaria la identidad del administrador, la cual puede ser obtenida mediante el bit pegajoso. Lo cual genera un hueco de seguridad que puede ser aprovechado con un desbordamiento de buffer para obtener privilegios del administrador [Smith05]. El modelo basado en permisos del sistema de archivos es también conocido como la versión mínima de la Lista de Control de Acceso (ACLs).

2.3.2. Modelo de Seguridad Multinivel

Los investigadores D. Elliott Bell y Leonard J. Lapadula propusieron el modelo de Seguridad Multinivel (MLS) [Bell73], el cual se fundamenta en una estructura de información tipo militar, en donde los sujetos se clasifican en niveles de confianza y los objetos en niveles de sensibilidad. Las etiquetas altamente secreto, secreto y no clasificado son empleadas en los niveles de clasificación de sujetos/objetos. Este modelo emplea el principio de la necesidad de conocer para limitar sólo el acceso a los objetos necesarios para que los sujetos realicen un trabajo especifico y así evitar el acceso a objetos con el mismo nivel de sensibilidad sin relación alguna. Dos propiedades definen la interacción entre sujetos y objetos, en donde L define un nivel de confianza o sensibilidad:

- *Propiedad Simple de Seguridad*: Un sujeto s puede tener acceso de lectura sobre el objeto o si y sólo si $L(o) \leq L(s)$ y el sujeto s tiene el permiso de leer al objeto o.

- *Propiedad estrella*: Un sujeto s puede tener acceso de escritura sobre el objeto o si y sólo si $L(s) \leq L(o)$ y el sujeto s tiene el permiso de escritura sobre el objeto o.

La Figura 2.5 muestra el flujo de datos en el modelo MLS para los niveles Altamente secreto, Secreto y No clasificado. La información importante que implica un riesgo de algún tipo si es divulgada es clasificada con la etiqueta correspondiente al nivel Altamente Secreto y la información que no representa un peligro si es divulgada se clasifica con el nivel de No clasificado. Un sujeto con la etiqueta "secreta" tiene permiso de escritura sobre los objetos con etiquetas de "altamente secreto", el permiso de lectura para sólo para objetos con etiqueta de igual o menor clasificación. El modelo MLS se caracteriza por la frase "sin escritura hacia abajo, sin lectura hacia arriba". El modelo MLS es un modelo del tipo de control de acceso obligatorio, en donde los usuarios pueden cambiar sus atributos de seguridad si y sólo si es permitido por la política de seguridad. El atributo de confidencialidad es la meta principal para el modelo MLS para prevenir la diseminación de información sensible.

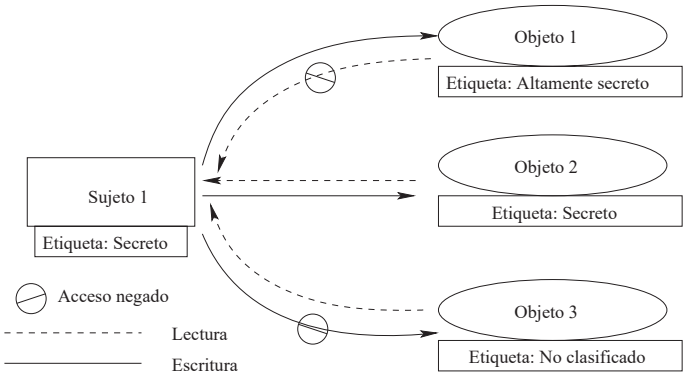

Figura 2.5: Flujo de Datos para el modelo MLS

2.3.3. Modelo de Integridad de Biba

Los investigadores Clark and Wilson [Clark87] hacen una comparación entre los sistemas de información militares y comerciales y concluyeron que es importante obtener

confidencialidad de la información para ambos ambientes, pero para los sistemas comerciales es necesario obtener la separación de tareas con el fin de obtener integridad sobre la información evitando con ésto los fraudes y errores debido al manejo individualizado.

El modelo de integridad propuesto por Biba [J.77] es similar al modelo MLS, pero la meta principal de éste es garantizar la integridad, previniendo modificaciones no autorizadas de información sensible. La Figura 2.6 muestra el flujo de datos para el modelo de integridad de Biba. Un sujeto con la etiqueta de "secreto" tiene permiso de acceso de lectura sobre los objetos con etiqueta "altamente secreto", el permiso de escritura sólo para objetos con etiquetas de clasificación igual o menor. El modelo de integridad de Biba es caracterizado por la frase "sin permiso de escritura hacia arriba, sin permiso de lectura hacia abajo". Este modelo corresponde al tipo de control de acceso obligatorio, al igual que el modelo de la muralla China [Bwever89], aplicado en ambientes comerciales.

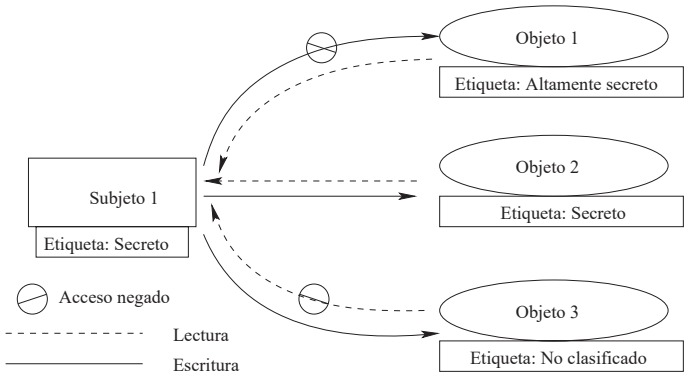

Figura 2.6: Flujo de Datos en el modelo de Integridad de Biba

2.3.4. Modelo de Cumplimiento de Tipo

La tecnología de cumplimiento de tipo (TE[1]) es una marca registrada de Secure Computing; en este esquema los procesos son confinados en celdas que permiten sólo el acceso a los recursos necesarios para realizar un cierto trabajo (privilegio mínimo).

El modelo de Cumplimiento de tipo y dominio (DTE[2]) [Boebert85], es una va-

[1]por sus siglas en Inglés de Type Enforcement
[2]por sus siglas en Inglés Domain and Type Enforcement

riante de TE, en donde los procesos son clasificados en dominios y los objetos en tipos. Las relaciones entre sujetos y objetos son especificadas por la tabla de cumplimiento de dominio y tipo (DTET), y la tabla de transisión de dominios (DTT). La Figura Figura 2.7 muestra la tabla DTET que especifica los tipos y los modos de acceso que pueden ser accedidos por los dominios. La Figura 2.8 muestra la tabla DTT que especifica las transiciones entre dominios.

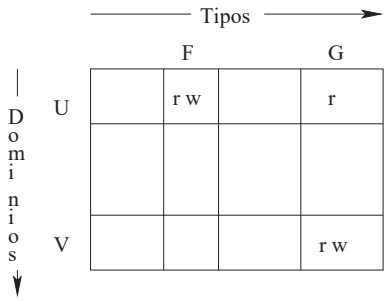

Figura 2.7: Tabla DTET

En el año de 1995 el modelo DTE fue propuesto para ser implementado en el sistema operativo UNIX [Badger95], lo cual fue realizado en el año de 1996 [Walker96] para confinar algunos procesos como httpd, el cual requería la identidad del administrador para su ejecución. En el sistema operativo Linux, el modelo DTE fue implementado en el año de 1997 por by Serge E. Hallyn [Hallyn97].

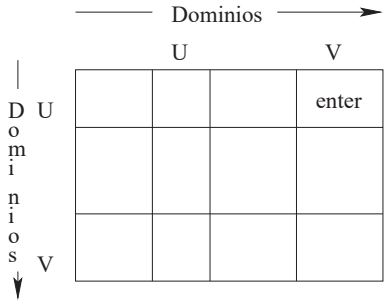

Figura 2.8: Tabla DTT

2.3.5. Modelo de Control de Acceso Basado en Roles

En el modelo del control de acceso basado en roles (RBAC[3]), un rol define tareas y responsabilidades de los miembros de una organización [Ferraiolo92]. Los roles doctor, enfermera, administrador son ejemplos de roles de un hospital. Recientemente el instituto nacional de estándares y tecnología ha propuesto crear un estándar para RBAC [Sandhu01] basado en el modelo de referencia y la especificación funcional de RBAC. El modelo de referencia de RBAC ofrece una rigurosa definición de los conjuntos y sus relaciones. La especificación funcional define los requerimientos para operaciones administrativas de creación y mantenimiento de los conjuntos y sus relaciones.

La Figura 2.9 muestre el modelo de referencia RBAC define a los conjuntos de usuarios-U, roles-R, sesiones-S, operaciones-OPS objetos-OBS y permisos-P. Un usuario es un humano, un rol es una ocupación dentro de una organización. Un permiso es un modo de acceso concedido a un usuario para utilizar un objeto u objetos. Los usuarios pueden obtener uno o más roles en diferentes sesiones.

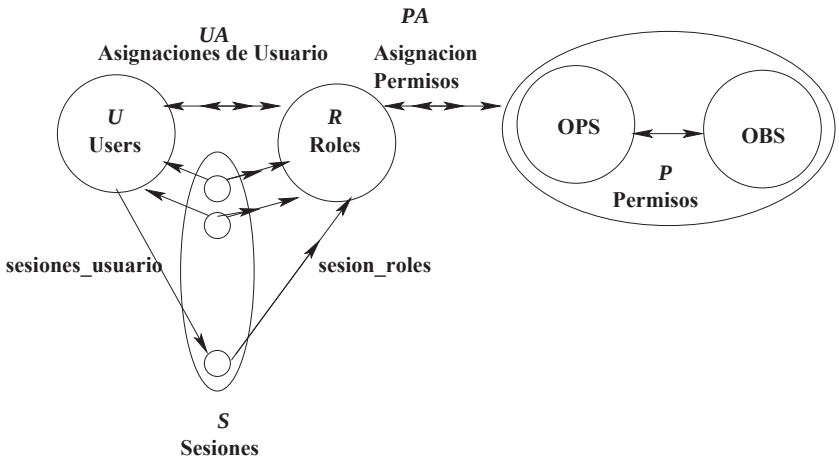

Figura 2.9: Núcleo RBAC

[3]por sus siglas en Inglés Role Based Access Control

2.4. Trabajos del Control de Acceso

Actualmente el control de acceso constituye una área extensa de investigación que ofrece muchos retos en el campo del diseño y desarrollo, esta conjuga a la líneas de la tecnología del control de acceso, modelos de control de acceso, análisis del control de acceso, colaboración segura, etc. A continuación se describen algunos trabajos de diferentes áreas del control de acceso.

2.4.1. Investigación sobre RBAC

En las grandes organizaciones el proceso de administración de los privilegios de acceso es manejado de una manera cooperativa entre los administradores distribuidos en diferentes localidades. Horst F.Wedde y Mario Lischka [F.Wedde03] proponen un método para la administración distribuida del modelo RBAC y especifica los requerimientos distribuidos para la administración.

Axel Kern, Andreas Schaad y Jonathan Moffett [Kern03] describen mediante la herramienta SAM Jupiter [Jupiter03] la administración de empresas que aplican al modelo acceso basado en rol.

Sylvia Osborn [Osborn02] emplea al modelo de grafos para roles [Nyanchama94], con la finalidad de obtener del flujo de información a partir de un grafo de roles. Sylvia L. Osborn, Yan Han y Jun Liu [Osborn03] proponen una metodología para administrar el modelo RBAC en un sistema de base de datos relacional, para esto se diseña una herramienta de administración de roles y grupos mediante grafos.

Una nueva propuesta para el manejo de restricciones temporales sobre el modelo RBAC, es planteada por James B. D. Joshi, Elisa Bertino, Basit Shafiq y Arif Ghafoor [Joshi03]. El nuevo modelo propuesto es nombrado GTRBAC, el cual permite especificar restricciones temporales en roles, asignaciones usuario-rol, asignaciones rol-permisos.

Jim Longstaff, Mike Lockyer y Jonh Nicholas [Longstaff03], presentan un nuevo modelo de autorización. Los autores aseguran que tal modelo es más poderoso que el modelo RBAC. El modelo es nombrado “Tees Confidentiality Model (TCM)”, la funcionalidad del

modelo está basada en la combinación del control de acceso basado en la identidad (IBAC) y algunas partes de RBAC. Este modelo fue desarrollado para ser utilizado en aplicaciones de salud del Reino Unido.

La asignación de roles a usuarios es una tarea que se realiza manualmente, mediante las reglas basadas en RBAC (RBRBAC) y los mecanismos se asignan dinámicamente los roles a los usuarios. Mohammad A. Al-Kahtani and Ravi Sandhu [Al-Kahtani03], exploran las discrepancias entre los roles inducidos y la existente jerarquía de roles.

Martin Kuhlmann y Gerhard Schimpf [Kuhlmann03], presentan un proceso para detectar mediante las minería de datos patrones de permisos de acceso contenidos en una base de datos con la finalidad de obtener roles del modelo RBAC.

Elisa Bertino, Barbara Catania y Maria Luisa Damiani [Bertino05], presentan al modelo GEO-RBAC, para la asignación de roles dependiendo de la localización de los usuarios en el espacio. Un agente de ventas que se encuentra camino a una ciudad destino, a este se le asigna el rol de viajero y asumirá otro rol cuando el usuario se llegue a la ciudad destino.

2.4.2. Trabajos relacionados con SELinux

Una herramienta nombrada Gokyo es presentada por Trent Jaeger, Reiner Sailer y Xiaolan Zhang [Jaeger04], la cual tiene la finalidad de solucionar conflictos generados por las restricciones en las reglas de SELinux. El trabajo de SELAC [Zanin04] desarrollado por Giorgio Zanin y Luigi Vincenzo Mancini, provee una especificación formal del control de acceso de SELinux usando una política arbitraria. Algunas herramientas han sido propuestas para el desarrollo, manipulación y análisis de políticas SELinux [Nakamura05], [tre05], [Herzog05] y [Zanin04].

2.4.3. Control de Uso

El control de uso (*usage control* UCON), es una nueva propuesta de control de acceso presentada por Jaehong Park y Ravi Sandhu [Park02]. UCON unifica las áreas de control de acceso tradicional, la administración de confianza (*trusted management*) y la administración de permisos digitales (*digital rights management*). El control de acceso tra-

dicional se concentra en sistemas cerrados donde todos los usuarios son conocidos y utiliza del lado del servidor al monitor de referencia dentro del mismo sistema. La administración de confianza es introducida para cubrir la autorización para estraños en un sistema abierto, tal como Internet. La administración de permisos digitales realiza el control del lado del cliente para el uso de información digital. El término "uso" significa uso de permisos sobre objetos digitales y el término "permisos" incluye los permisos de uso para objetos digitales y permisos para delegación de permisos.

El modelo UCON, mostrado en la Figura 2.10 consiste de un componente esencial y adicional. El componente esencial consiste de sujetos, objetos y permisos. El componente adicional consiste de reglas de autorización, condiciones y obligaciones.

Sujetos

Los sujetos son entidades con atributos asociados y ejercen ciertos permisos sobre los objetos. Un sujeto puede ser un usuario, un grupo, un rol o un proceso. Algunos ejemplos de atributos son las identidades, roles, créditos, miembros y niveles de seguridad.

Objetos

Los objetos son entidades que mantienen permisos por parte de los sujetos o bien los sujetos pueden tener acceso o uso de los objetos. Los objetos también tienen atributos asociados, tal como los niveles de seguridad, propietarios, clases, etc.

Permisos

Los permisos son privilegios que un sujeto tiene sobre un objeto. Los permisos son un conjunto de funciones de uso, las cuales permiten el acceso por parte de los sujetos sobre los objetos.

Reglas de Autorización

Las reglas de autorización son un conjunto de requerimientos que deben ser satisfechos antes de permitirle a los sujetos el acceso o uso de objetos.

Condiciones

Las condiciones son un conjunto de factores de desición que el sistema debe verificar durante el proceso de autorización junto con las reglas de autorización antes de permitir el uso de permisos sobre objetos digitales.

Obligaciones

Las obligaciones son los requerimientos obligatorios del sujeto y deben ser verificados antes que éste obtenga o ejerza permisos sobre un objeto.

El proyecto PCASSO desarrollado por la Universidad de San Diego y el SAIC [Baker97], es una aplicación del modelo UCON, el cual intenta ofrecer el acceso seguro desde Internet a la información altamente sensible de pacientes para el dominio de un hospital.

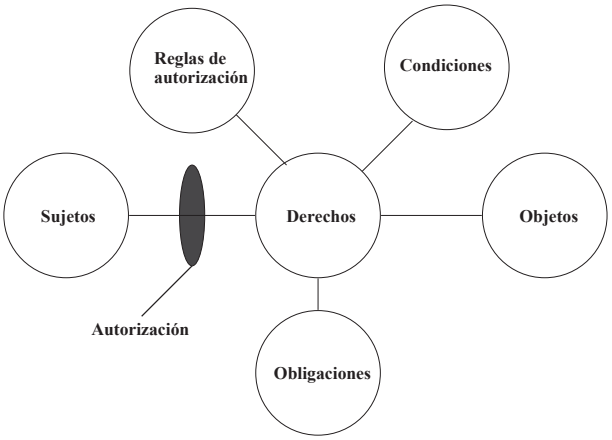

Figura 2.10: Componentes del modelo UCON

Xinwen Zhang, Jaehong Park y Francesco Parisi Presicce [Zhang04], presentan la especificación del modelo UCON empleando lógica de acción temporal de Lamport.

2.4.4. Sistemas Cooperativos

Los investigadores Charles E. Phillips y Steven A. Demurjian [Phillips02], plantean al problema dinámico de la coalición (DCP), presentado en situaciones de crisis (desastres

naturales, asistencia humanitaria, incidentes internacionales, guerra, operaciones de combate de guerra) nacionales o internacionales. El problema de la coalición dinámica, se define como la seguridad inherente, recurso y/o el riesgo de compartir información y recursos al ocurrir una coalición necesaria para la resolución de una crisis. Se propone una red entre las coaliciones basada en la tecnología de cortafuego y una base de datos combinada en la que compartan información los cuerpos de inteligencia y militares. El problema de la coalición dinámica se agudiza al intervenir en la resolución del problema organizaciones civiles, por el hecho del manejo de información sensible por parte de un gran número de participantes que constantemente están cambiando.

Es un problema interesante el desarrollo de infraestructura para redes para el intercambio seguro de información, así como las diferentes políticas para la administración del intercambio. Una familia de modelos de control de acceso basados en coaliciones (CBAC) [Cohen02] es presentada por Even Cohen, William Winsborough y Roshan K.

2.4.5. Administración de Políticas de Control de Acceso

Los investigadores Trent Jaeger, Anthony Edwards y Xiolan Zhang [Jaeger02], plantean el concepto del espacio para el control de acceso, con la finalidad de utilizarlo en la administración de políticas de control de acceso. Los permisos asignados a un sujeto son divididos en los subespacios especificado, prohibido, permisible, desconocido y obligado. Los subespacios están basados en las asignaciones y restricciones de la política. El subespacio especificado, contiene los permisos asignados en la configuración actual, el subespacio prohibido contiene los permisos asignados a las restricciones. El subespacio permisible, contiene al conjunto de permisos que pueden ser otorgados. El subespacio de permisos desconocidos contiene la parte del espacio que no se encuentra entre los subespacios permisible y prohibido. El subespacio obligado, es requerido para garantizar el funcionamiento correcto del sistema. Un ejemplo ideal del espacio para el control de acceso, se muestra en la Figura 2.11. Se observa que no se tienen subespacios en conflicto, por la ausencia de traslape entre los subespacios obligado, especificado, permisible, desconocido y prohibido. La Figura 2.12, muestra un ejemplo realista en la que se tienen subespacios en conflicto. Una parte de subespacio obligado se traslapa con el subespacio desconocido y el subespacio especificado

se traslapa con los subespacios desconocido y prohibido. Se dice que el espacio es completo para la política, cuando la combinación permisible y prohibido eliminan al subespacio desconocido. Cuando los subespacios permisible y especificado son iguales se garantiza el privilegio mínimo. Para los administradores es una tarea ardua el eliminar los subespacios en conflicto y para esto se plantea el desarrollo de la herramienta nombrada Gokyo.

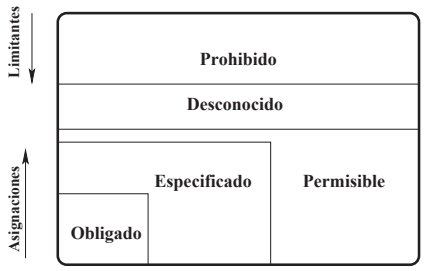

Figura 2.11: Espacio ideal de control de acceso

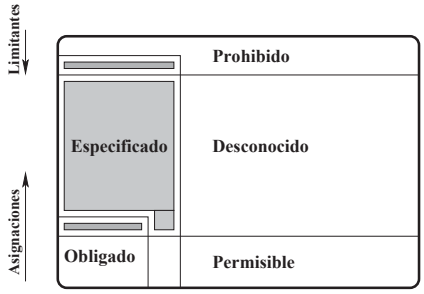

Figura 2.12: Espacio real de control de acceso

Marina Bykova y Mikhail Atallah [Bykova04], proponen el uso de cadenas bits para representar políticas. Los usuarios y los documentos tienen asociado una política representada por cadenas de bits. Un usuario tiene acceso a un documento si el valor de los bits de su política es mayor o igual que la política asociada al documento.

2.4.6. Aplicaciones del Control de Acceso

El proyecto PERMIS desarrollado por David W. Chadwick y Alexander Otenko [Chadwick02], plantea una infraestructura para administrar roles de usuario mediante un

atributo para el certificado digital X509. Para esto, se emplea al estándar ITU-T X.509, con la finalidad de aprovechar las técnicas fuertes de autenticación basadas en firmas digitales, certificados digitales e infraestructura de clave pública. La infraestructura propuesta, tendrá las funciones de autenticar al usuario y autorizar las operaciones solicitadas. El atributo del certificado digital correspondiente al rol, contiene la información del sujeto y las especificaciones del rol asignado en lenguaje XML. Una de las aplicaciones para esta propuesta es el intercambio de información entre cliente - servidor.

Para la administración de cierta organización los usuarios son divididos en clases de seguridad, en donde las clases especifican a que tienen acceso los usuarios. Una solución basadas en claves criptográficas genera para cada nivel de seguridad una clave, Indrakshi Ray, Indrajit Ray y Natu Narasimhamurthi [Ray02] proponen una solución para soportar jerarquía mediante un esquema similar al criptosistema RSA.

Una política para localización de personas con dispositivos moviles es propuesta por Urs Hengartner y Peter Steenkiste [Hengartner04]. La información de localización tendrá asociada las propiedades granularidad, localidades e intervalos de tiempo. La autorización y autenticación esta basada en certificados SPKI/SDSI.

El control de acceso criptográfico para garantizar la integridad y confidencialidad de datos almacenados en un servidor es propuesto por Anthony Harrington y Christian Jensen [Harrington03]. La finalidad del trabajo es ofrecer un sistema de archivos distribuido que permita transferir datos desde lugares inseguros como Internet, utilizando criptografía de clave pública.

2.4.7. Administración de Sistemas

Oerg Abendroth y Christian D. Jensen [Abendroth03b], proponen cambiar del paradigma “el control de acceso administrado centralmente y solemnemente por la compañia” al paradigma que permita la participación de empresas especializadas en seguridad. Exponen las siguientes razones:

- Administración compleja del sistema.

- Bajo rendimiento de empleados locales en seguridad.

Además de argumentar que el costo de los servicios externos puede ser amortizado por varias compañias. Para permitir la participación externa, proponen un mecanismo de control de acceso basado en la estructura ASCap [Abendroth03a].

2.5. Trabajo Relacionado

El proyecto prototipo para administración y distribución de políticas SELinux (PDM) [pdm08], desarrollado por Tresys Technology esta formado por los componentes servidor, agentes y una consola de administración. El servidor de PDM es un proceso que tiene como meta la administración y distribución de políticas de red.

Las funcionalidades actuales del servidor PDM:

- centralización y autorización de política fuente
- clasificación jerárquica de los sistemas administrados.
- Distribución inteligente de política: actualizando sólo a los sistemas necesarios.
- Posibilidad de incorporar auditoría.
- Control básico para el control de revisión de la política.

Este proyecto tiene sus orígenes en el año 2008 y en el futuro se pretende agregarle una infraestructura para la administración tal como FreeIPA (Identity, Policy, Audit), la cual combina a la distribución Fedora de Linux, un servidor de directorios de Fedora, a Kerberos, NTP[4] y DNS[5]. Ofreciendo una interfase web y de línea de comandos con la finalidad de administrar identidades para el soporte de política y auditoría.

Este proyecto se encuentra en la fase de inicio, no se tiene evidencia de algunos problemas encontrados tales como la co-dependencia entre roles y dominios, ni de las condiciones para la actualización de políticas.

[4]del Inglés *N*etwok *T*ime *P*rotocol
[5]del Inglés *D*omain *N*ame *S*ystem

2.6. Conclusiones

Los modelos de control de acceso tradicional presentados en este capítulo emplean al factor *cualitativo*, el cual especifica a que objetos se podrá tener acceso. El factor *cuantitativo* es un complemento para el factor cualitativo, el cual especifica el número de acceso a los objetos. Al incluir al factor cuantitativo en el control de acceso tradicional se podría eliminar al *mal uso*[6] de recursos ocasionado posiblemente por gusanos, virus, troyanos, etc.

El control de acceso es un área de investigación activa desde los años sesentas, diferentes modelos de control de acceso han sido propuestos y sólo unos cuantos se han implementado. Considero que SELinux es una de las contribuciones más importantes para está área, ya que conjuga dos tipos de control de acceso y cuatro modelos de control de acceso, además de requirir y poseer una infraestructura operativa y administrativa. Desde mi punto de vista personal considero que el sistema operativo debe ser la base principal para la seguridad de la información, de tal manera que se minimice la participación del espacio de aplicación para este fin.

[6]del Inglés *misuse*

Capítulo 3

SELinux

Según Silberschatz [Silberschatz05], un sistema operativo debe proveer ciertas funcionalidades tales como administrar hardware, ofrecer una base para los programas de aplicación, ofrecer protección y seguridad, actuar como intermediario entre el hardware de la computadora y el usuario.

Los usuarios u organizaciones pretenden obtener de todo sistema operativo los atributos de *integridad*, *confidencialidad* y *disponibilidad* tanto en los recursos de hardware como en software, aunado a una fácil administración y sencilla operación para los usuarios. Para resolver un gran número de problemas de seguridad los investigadores y desarrolladores han diseñado y desarrollado sistemas operativos que fortalecen el control de acceso del kernel [Loscoco00]. Un ejemplo de este tipo sistema operativo es Linux que integra al control de acceso de SELinux (del inglés Security-Enhanced Linux), en el cual algunos modelos de control de acceso obligatorio son implementados a nivel del kernel.

En la Figura 3.1, se representa al control de acceso aplicado en los sistemas operativos. Para lo cual los procesos necesitan tener acceso a los recursos y para ello el kernel es fortalecido con una base de datos correspondiente a un conjunto de reglas (política). Una principal característica para el mecanismo de la base de datos del control de acceso es el permitir que las reglas puedan cambiar dinámicamente.

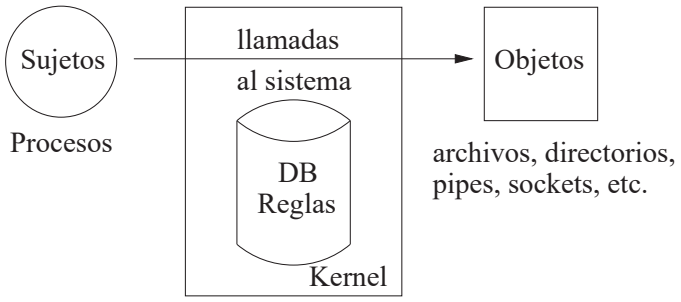

Figura 3.1: Control de Acceso

3.1. Descripción

SELinux fue desarrollado por la Agencia de Seguridad Nacional (NSA) de los Estados Unidos [sel09]. SELinux está diseñado para soportar una amplia variedad de políticas de seguridad, de tal manera que una política SElinux consiste de un conjunto de reglas basadas en cuatro diferentes modelos de control de acceso; Cumplimiento de Tipo [typ09] [Badger95], Seguridad Multinivel [Ferraiolo92], una parte del modelo Basado en Roles [Ferraiolo92] y el modelo de Identidad de Usuario.

El kernel flux de seguridad avanzada (*Flask*) [R.99] es una arquitectura flexible que soporta varias políticas obligatorias. La arquitectura de seguridad Flask fue diseñada para ambientes de microkernel, la cual fue insertada en el kernel monolítico de Linux con el nombre de SELinux [Loscoco01]. La Figura 3.2, muestra los componentes principales del módulo de SELinux formado por un sistema de archivos que tiene la finalidad de proveer la interacción entre los espacios de usuario y del kernel, un vector cache de acceso para registrar los accesos más frecuentes y el servidor de seguridad que contiene a las reglas de la política SELinux.

3.2. Operación

En SELinux, el control de acceso discrecional interactúa con el control de acceso obligatorio. En la Figura 3.3 se muestra un sujeto que solicita el acceso a un recurso, SELinux analiza la decisión de control DAC y si este niega el acceso entonces SELinux niega el acceso

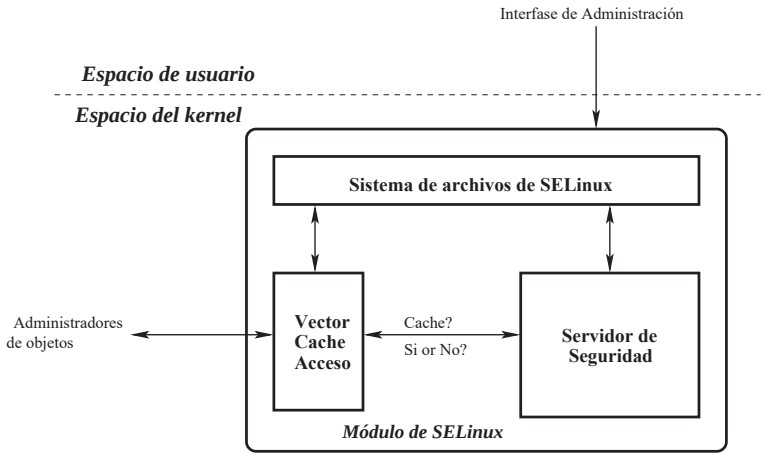

Figura 3.2: Arquitectura del Módulo de SELinux

al recurso. Si el control de acceso DAC permite el acceso al recurso, entonces se analiza la desición del control MAC para negar u otorgar el acceso.

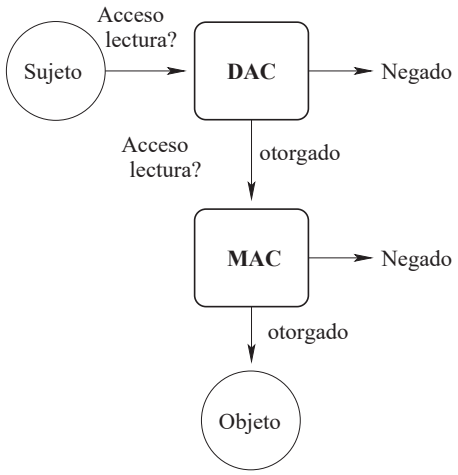

Figura 3.3: Interacción entre DAC y MAC

La Figura 3.4, muestra la interacción entre un sujeto y un objeto mediante el control de acceso discrecional. En el sistema operativo Linux tradicional, la lista de control de acceso es el modelo usado para el control de acceso discrecional. Este modelo está basado en la identidad del sujeto y emplea al conjunto de permisos para el propietario, grupo, y

otros, respectivamente.

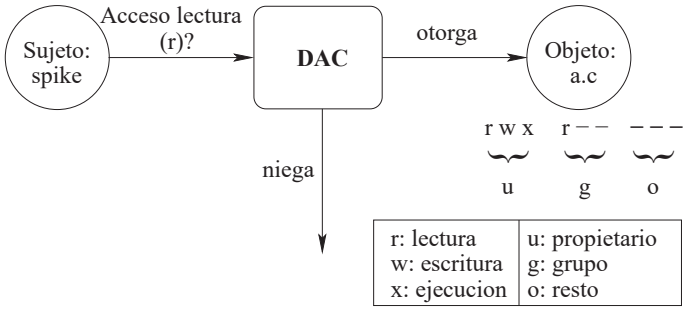

Figura 3.4: Control de acceso discrecional con modelo basado en permisos

3.2.1. Contexto de Seguridad en SELinux

En SELinux los sujetos y objetos tienen asociado un conjuto de atributos de seguridad nombrado el contexto de seguridad (Scontext), el cual se basa en los modelos de control de acceso utilizados. En la Figura 3.5 muestra la etiqueta de seguridad que es formada por el atributo identidad, rol, tipo y nivel tomados de los modelos identidad de usuario, rbac, cumplimiento de tipo y multinivel de seguridad respectivamente.

Scontext	**user_u**	**system_r**	**system_t**	**s0**
Atributo	identidad	rol	tipo	nivel
Modelo	UI	RBAC	TE	MLS

Figura 3.5: Contexto de seguridad para sujetos y objetos

La Figura 3.6, muestra la interacción entre los diferentes atributos de seguridad de SELinux. Un usuario con un atributo de *Identidad* puede tener acceso a uno ó más atributos de *rol*, un atributo de rol puede tener acceso a uno ó más atributos de *dominio*, y un atributo de dominio combinado con el atributo de *nivel* puede tener acceso a uno ó más atributos de *tipo* con valores del atributo nivel menores o igual al del atributo dominio. Otro aspecto importante es la dominancia entre roles que permite establecer la jerarquía de roles y las transiciones entre dominios que le permite a un dominio adoptar las propiedades de otro dominio [Badger95].

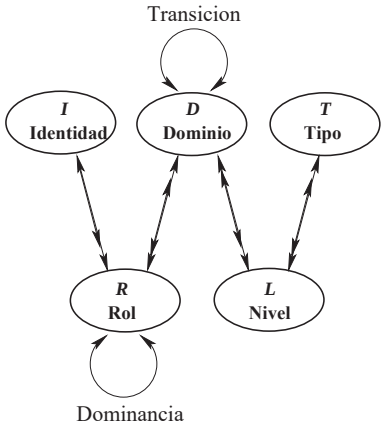

Figura 3.6: Interacción entre atributos

La Figura 3.7, muestra la interacción entre un sujeto y un objeto mediante el control de acceso obligatorio. Este tipo de control de acceso analiza los contextos de seguridad del sujeto y objeto. El sujeto tendrá acceso de lectura para el objeto *a.c* si y solo si existe una regla que permita el acceso a los objetos del tipo *test_t* por parte del atributo *system_t* y el atributo de nivel de seguridad del objeto es menor o igual al nivel de seguridad del objeto. Otra posibilidad de acceso se puede dar si el atributo rol *system_r* domina a un rol que tiene el acceso a un atributo que puede tener acceso a los objetos con el atributo del tipo *test_t* y el criterio del nivel es cumplido. En SELinux el atributo de rol object_r es aplicado a los recursos del sistema.

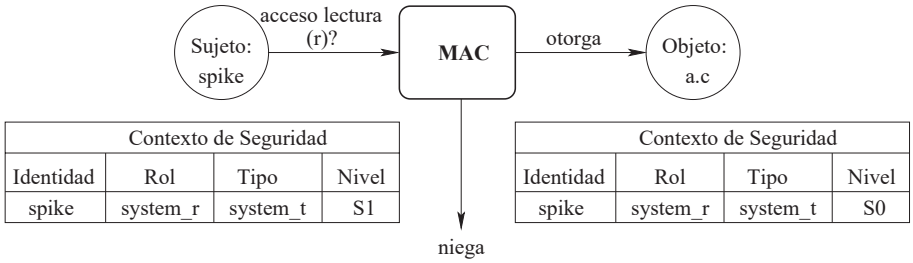

Figura 3.7: Control de acceso obligatorio (MAC) en SELinux

3.3. Políticas

Una política SELinux consiste de un conjunto de reglas basadas en diferentes modelos de control de acceso. Actualmente, las políticas SELinux son administradas localmente, en donde las reglas son especificadas en archivos agrupados en varios directorios. SELinux define dos tipos de políticas; política estricta y por objetivos. Inicialmente SELinux fue diseñado para emplear políticas estrictas, en donde las reglas por default niegan todo acceso y se emplean reglas para otorgar el acceso. Por razones de estrategia los desarrolladores de SELinux construyeron la política por objetivos en donde las reglas permiten todo acceso y se emplean reglas para negar el acceso. Más información sobre política estricta y por objetivo puede consultar [Walsh03] [Walsh05].

La Figura 3.8, muestra a la estructura de directorios empleados para almacenar políticas locales de SELinux. Uno de estos directorios es asociado a las clases, objetos y las estructuras de permisos definidas por la arquitectura *Flask*. Similarmente, dos directorios son asociados a las reglas basadas en los modelos TE y RBAC. El directorio asociado con las reglas de RBAC es combinado con el directorio para las declaraciones de usuario. El directorio para dolas restricciones contiene al conjunto de reglas empleadas para negar otras reglas. Por ejemplo, si existe una regla que permita el acceso a un recurso, puede existir una restricción que invalide tal regla, prohibiendo el acceso al recurso. El directorio del contexto de seguridad especifica el contexto de seguridad de cada elemento de la estructura de archivos y directorios.

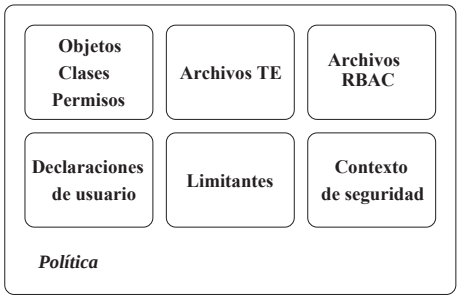

Figura 3.8: Estructura de directorios de una Política SELinux

La Figura 3.9, muestra la construcción de una política SELinux, la cual es agrupada en un conjunto de archivos que especifican clases, permisos, usuarios, roles, tipos, dominios, niveles, etc. Estos archivos son concatenados y procesados por el preprocesador m4[1] para reemplazar las macros por las reglas básicas de SELinux y producir un archivo especificado en el lenguaje de ensamble de política. Este nuevo archivo es utilizado en el proceso de compilación para producir un archivo que contenga la política especificada en el formato del lenguaje máquina de política, el cual es utilizado para ser insertado en el servidor de políticas del kernel mediante el sistema de archivos de SELinux. En el Apéndice A, se muestran las estructuras básicas de la política SELinux.

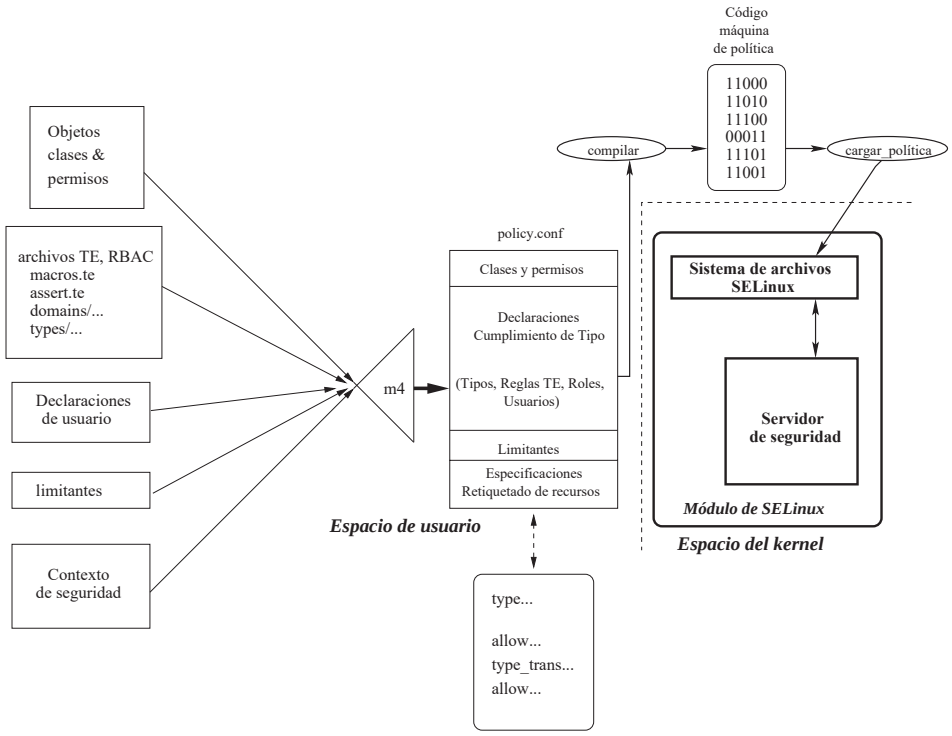

Figura 3.9: Compilación, verificación y carga de Políticas SELinux

[1]Implementación del tradicional macro procesador de Unix

3.4. Conceptos de SELinux

A continuación se definen los aspectos básicos y las directivas principales de SELinux.

1. Aspectos básicos de SELinux.

Las clases de objetos, la identidad, el rol, nivel, tipo y dominio constituyen los aspectos básicos de SELinux, ya que el contexto de seguridad para objetos y sujetos se basa en tales aspectos.

Clases

Las clases representan categorias para los recursos, tales como:
Archivos, Sockets, Sistemas de archivo, Procesos, Enlaces, Directorios, Descriptores de archivo, etc.

Una clase de objetos define un conjunto de permisos o acciones que los sujetos pueden ejercer sobre los objetos de tal clase. Por ejemplo, para la clase archivo (file) se tienen los siguientes permisos:
Append, Create, Execute, Get attribute, Link, Lock, Read, Rename, Write, etc.

Identidad

Cada usuario (sujeto) tiene asociado una identidad que corresponde normalmente a su nombre.

Rol

Los miembros de una organización pueden desarrollar varias actividades, las cuales están basadas en sus actividades encomendadas. El atributo de rol es usado para clasificar las funciones asignadas a los usuarios. Dependiendo de las actividades ha desarrollar los usuarios pueden emplear uno o más roles.

Dominio y Tipo

Los procesos se agrupan en dominios, y los recursos del sistema (objetos) se agrupan en

tipos. Un dominio puede tener acceso a uno ó más tipos para un conjunto especifico de acciones y la transición entre dominios permite que un dominio puedan adoptar las propiedades de otro dominio, de acuerdo con el modelo de Cumplimiento de Tipo.

Nivel

El atributo de nivel es asignado a sujetos y objetos de tal forma que un sujeto s puede tener acceso a un objeto o para un cierto conjunto de acciones, si y solo si el atributo de nivel del sujeto s es mayor o igual que el nivel del objeto o.

2. Principales Directivas de SELinux.

A continuación se definen algunas directivas para las reglas basadas en modelos TE y RBAC.

2.a Reglas TE.

Las reglas TE definen atributos, tipos, dominios, transiciones entre dominios y vectores de acceso.

Atributo

Un atributo define una propiedad para los tipos, un tipo puede tener cualquier cantidad de atributos. Algunos ejemplos de atributos:

file_type. Este atributo es para todos los tipos asignados a los archivos.

fs_type. Atributo que identifica a todos los tipos asignados para los sistemas de archivo, incluyendo a los sistemas de archivos no persistentes.

exec_type. Atributo que permite agrupar a todos los tipos que son asignados para ser ejecutados.

domain. El atributo domain identifica a cada tipo que es un proceso.

Tipos

El atributo tipo es empleado para agrupar los recursos del sistema y dominios para los

procesos. Un tipo con el atributo domain define un dominio que confina un proceso. Regla para declaración de tipos:
type <type_name> [aliases] [atributes]

Transición de dominios

La transición de dominios permite cambiar el dominio de un proceso que se ejecuto a un dominio diferente. Regla para declaración de transición de tipos:
type_transition <source_type(s)> <target_type(s)> <class(es)> <new_type>

Vectores de acceso

Los vectores de acceso son reglas que le permiten a los dominios el acceso a varios objetos. Un vector de acceso consiste de un sujeto, objeto, clase o clases y los permisos para el sujeto:

<av_kind> <source_type(s)> <target_type(s)>:<class(es)> <permission(s)>
Un tipo de vector de acceso av_kind puede definirse como:

allow. Permite a un sujeto actuar de una manera especifica sobre un objeto.
allow named_t sbin_t:dir search;

auditallow Registra en bitácora el acceso otorgado.
auditallow unconfined_t security_t:security{load_policy setenforce};

dontaudit. No registrar en bitácora la negación de acceso.
dontaudit named_t root_t:file {getattr read};

neverallow. Permite definir vectores de acceso que no serán permitidos.
neverallow domain ~domain:process transition;
(~domain significa no ser dominio)

2.b Reglas para usuarios y roles.

Roles

Un rol define que dominios puede tener acceso una identidad de usuario. Reglas para definir

roles:

role <rolename> types <domain(s)>;

Se puede definir la extensión de roles.

allow <rolename(s)> <new_role(s)>;

La dominancia es otra posibilidad para la extensión de roles.

dominance <role_master> <role(s)_dominated>;

Usuarios

Cada identidad de usuario puede tener acceso a un conjunto de roles. Regla para definir usuario y roles:

user username roles role_set [ranges MLS_range_set];

Algunos identificadores de roles son empleados en las políticas estricta y por objetivos, tales como:

system_r. Este rol es empleado para agrupar los dominios correspondiente a todos los procesos excepto los procesos de usuario.

user_r. Este rol estándar para los usuarios regulares de Linux.

object_r. Rol aplicado a todos los objetos correspondientes a los recursos del sistema.

sysadm_r. Rol administrativo para una política estricta.

Para la política por objetivos todos los sujetos son confinados en el dominio *unconfined_t*, y solo algunos procesos son confinados en diferentes dominios; Para este mismo tipo de poítica, los usuarios tienen asociado la identidad *user_u*. A diferencia de la política estricta, en donde los sujetos tienen asignado diferentes identidades y los procesos son confinados en diferentes dominios.

3.5. Modelo del Control de Acceso de SELinux

El trabajo SELAC [Zanin04] consiste de una descripción formal del control de acceso de SELinux, el cual emplea a los principales constructores que definen al problema

de accesibilidad de sujetos a objetos. Una política arbitraria es utilizada para obtener los conjuntos que permiten determinar cuando un sujeto puede tener a un recurso mediante un cierto modo de acceso. A continuación se muestra parte de tal modelo, el cual se construye del análisis de los diferentes archivos de configuración que integran la política SELinux.

Conjuntos básicos de SELAC
Cada uno de los conjuntos del modelo son generados a partir de una política arbitraria de SELinux.

El conjunto C tiene como elementos a todas la clases de objetos.

- $C = \emptyset$
- Para cada regla del tipo *class* (c) contenida en los archivos de configuración:
  - $C = C \cup \{c\}$

Un vector de acceso es definido para cada clase de objetos. El conjunto *P(c)* tiene como elementos a todos los modos de acceso de una clase de objetos c.

1. $P(c) = \emptyset$
2. Para cada regla del tipo *access_vector* $(c, p_1, ..., p_n)$:
   - $P(c) = P(c) \cup \{(p_1, c),...,(p_n, c)\}$
     donde $p_1, ..., p_n$ son permisos o acciones.

El conjunto A tiene como elementos a todos los atributos para los sujeto/dominio u objeto/tipo:

1. $A = \emptyset$
2. Para cada regla del tipo *attribute* (a):

- $A = A \cup \{a\}$

El conjunto T tiene como elementos a todos los dominios y tipos:

1. $T = \emptyset$
2. Para cada regla del tipo *type* (t):
   - $T = T \cup \{t\}$
3. Para cada regla del tipo *type* $t, a_1, ..., a_n$,
   - para $a_1, ..., a_n \in A$: $T = T \cup \{t\}$

El conjunto R tiene como elementos a todos los roles:

1. $R = \{object\_r\}$
2. Para cada regla del tipo *role* $(r, t_1, ..., t_n)$:
   - $R = R \cup \{r\}$

$D(r)$ define al conjunto de los roles dominados por r:

1. $D(r) = \emptyset$
2. Para cada regla del tipo *dominance* (i, j):
   - para r, $j \in$ R: $D(r) = D(r) \cup j$

El conjunto U tiene como elementos a todos los usuarios:

1. $U = \emptyset$
2. Para cada regla del tipo $useru, r_1, ..., r_m$:

- $U = U \cup u$

El conjunto Cs tiene como elementos a todos los posibles contextos de seguridad para los sujetos y objetos:

$$Cs = \{(\text{u,r,t}) \mid \text{u} \in \text{U}, \text{r} \in \text{R}, \text{t} \in \text{T}\}$$

El conjunto Cv tiene como elementos a todos los posibles contextos válidos de seguridad para los sujetos y objetos:

$$Cv = \{(\text{u,r,t}) \in Cs \mid \text{u} \in \text{U}, \text{r} \in \text{R}, \text{t} \in \text{T}\}$$

El conjunto para los elementos correspondientes a los contextos de seguridad de los objetos de clase c:
$Cs(c) = \{(\text{u},\textit{object\_r},\text{t}) \in Cs \mid \text{u} \in \text{U}, \text{t} \in \text{T}\}$

La matriz de acceso TE es asociada al conjunto $M(s)$, en donde $s \in Cv$ correspondiente a un contexto válido de seguridad para un sujeto. Las restricciones son asociadas a la función $\gamma(s, o, p)$, y la función $\Delta(s, c)$ define al espacio de permisos autorizados para un contexto de seguridad s sobre una clase c. Este modelo es presentado más ampliamente en el Apéndice B.

3.6. Políticas de ejemplo

SELinux confina a los procesos en cajas de arena (ambientes de privilegio mínimo) limitando su acceso solo a los recursos necesarios para realizar su trabajo. Por ejemplo, el proceso del servidor web *apache* puede ser confinado para tener acceso a los recursos de configuración y de páginas web. Entonces el control de acceso solo permitirá acceso a los recursos necesarios por el dominio del proceso apache y negará el acceso al resto de los recursos. Las siguientes reglas son parte del conjunto de reglas que confinan al dominio del

proceso apache:

- Definición de reglas para el tipo apache_t con el atributo de dominio.
  - type apache_t, domain;
  - role apache_r types apache_t;
  - type apache_exec_t, file_type, exec_type;
  - allow apache_t web_resource_type : file {write read};

En tales reglas se define al rol *apache_r*, un dominio *apache_t*, y un tipo *apache_exec_t*. El dominio apache_t puede tener acceso de lectura y escritura sobre los archivos del tipo *web_resource_type*.

Una ejemplo más completo se muestra en el siguiente fragmento de código escrito en lenguaje C, el cual se escribio para confinar a un proceso en un dominio nombrado *pid_t*. En este código se ejecutan varias funciones para obtener la información del proceso padre e hijo, además de obtener algunos parámetros del sistema operativo. Los resultados de tales funciones son impresas en la salida estándar. En las reglas necesarias para el dominio pid_t se observa la regla que 12 permite tener acceso de lectura y obtención de atributos para los objetos de clase file y tipo etc_t. Esto se debe a la compilación estática del código, y por esta razón el proceso requiere leer las librerias.

Código en C

```
Linea
0 /*pid.c (fuente) pid (destino)*/
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/utsname.h>

4 int main(int argv, char **argc)
5 {
6  int i;
7  struct utsname buf;
8  printf("The process ID is %d\n",(int) getpid());
9  printf("The parent process ID is %d\n",(int) getppid());
10  i = uname(&buf);
11  printf("The Operating System is %s\n",buf.sysname);
12  printf("node name: %s\n",buf.nodename);
13  printf("release: %s\n",buf.release);
14  printf("version: %s\n",buf.version);
15  printf("machine: %s\n",buf.machine);
16  #ifdef _GNU_SOURCE
17  printf("domainname: %s\n",buf.domainname);
18  #endif
19  return 0;
20 }
```

Reglas para el dominio pid_t

```
Linea
1 #pid.te
2 #DESC pid command
3 type pid_t, domain; # se declara un dominio pid_t
4 type pid_exec_t, file_type, exec_type; # se crea un tipo pid_exec_t con
5 role sysadm_r types pid_t; # se permite al rol sysadm_r usar al dominio pid_t
6 domain_auto_trans(sysadm_t, pid_exec_t, pid_t); # transici\'on del dominio sysadm_t al dominio pid_t
7 # al ejecutar al tipo pid_exec_t
8 allow pid_t sysadm_devpts_t: chr_file { read write getattr }; # read write getattr name="0"
9 allow pid_t newrole_t: fd { use }; # use name="0"
10 allow pid_t root_t: dir { search }; # search name="/"
11 allow pid_t etc_t: dir { search }; # search name="etc"
12 allow pid_t etc_t: file { read getattr }; # read getattr name="ld.so.cache"
13 allow pid_t lib_t: dir { read getattr search }; # read getattr search name="lib"
14 allow pid_t lib_t: lnk_file { read }; # read name="libc.so.6"
15 allow pid_t shlib_t: file { read getattr execute }; # read getattr execute name="libc-2.3.5.so"
16 allow pid_t device_t: dir { search }; # search name="dev"
17 allow pid_t sysadm_tty_device_t : chr_file { read write getattr ioctl }; # read write getattr ioctl name="tty1"
18 allow pid_t local_login_t : fd { use }; # use name="tty1"
19 allow pid_t null_device_t : chr_file { getattr write }; # getattr write name="null"
```

El cumplimiento de tipo constituye la parte más importante del control de acceso de SELinux. El 99 por ciento de las reglas de una política se basan en TE y de manera general una política establece en promedio 50,000 reglas para las declaraciones de tipos, atributos, alias, vectores de acceso, transiciones, etc [typ05] [ter05]. Una de las ventajas principales de SELinux es la alta granularidad para los permisos, ya que define 55 clases de objetos y 197 permisos.

El diseño, configuración, análisis y administración de políticas de seguridad de SELinux son tareas complejas para los administradores de sistema. Los administradores de sistema SELinux necesitan conocer los modelos de control de acceso respectivos y los pasos correctos de configuración. Las macros m4 y los modelos de control de acceso son requisitos indispensables para escritura de políticas de SELinux.

Una mayor referencia de SELinux puede encontrarla en [Frank Mayer06, McCarty04, Red05, Stephen200505, Coker05, Cokerrg], eventos de SELinux [SELinux06], y un curso para ingenieros SELinux desarrollado por Mitre, esta disponible en la página: http:\\lsc.fie.umich.mx\~pedro\selinux-course-outline.tbz.

3.7. Conclusiones

En los inicios de SELinux operaba con la política estricta y por razones de continuidad fue necesario desarrollar la política por objetivos, lo cual implico cambios significativos para la operación y administración del sistema operativo. Un aspecto importante que no ha beneficiado a SELinux es la falta de información actualizada para la operación y administración aunado a un desconocimiento de los modelos de control de acceso empleados en éste. De tal manera que es necesario cubrir con estos dos aspectos para proyectar de una manera más completa a SELinux.

Capítulo 4

Arquitectura para la Administración de Políticas SELinux en Ambientes Distribuidos (A^2PSAD)

Cuando las políticas SELinux en un ambiente distribuido sufren cambios, los administradores deben realizar las operaciones de modificación y actualización de políticas en todos los equipos involucrados. Para esto se deben emplear prácticas que permitan obtener *confidencialidad* e *integridad* en los procesos de modificación y actualización de políticas.

El presente trabajo plantea una arquitectura que proporciona una base sólida para la administración de políticas SELinux aplicadas en ambientes distribuidos; tal arquitectura proporciona un manejo sencillo y seguro para las operaciones de modificación y actualización. De tal manera que para ambientes distribuidos se minimice la posibilidad de inseguridad en los procesos administrativos de políticas SELinux. Esta arquitectura se conforma por un servidor de políticas y clientes de políticas. El servidor de políticas centraliza las políticas del ambiente distribuido y las segmenta en políticas individuales para los clientes de políticas.

El atributo localidad es utilizado para denotar a cualquier computadora del am-

biente distribuido. Con este atributo se crean extensiones de regla que permiten especificar una relación localidad-roles. Esta relación define el conjunto de roles que pueden ejercer los usuarios en una localidad l. Una relación usuario-localidad-roles define el conjunto roles que puede ejercer un usuario u en la localidad l. Un servidor de políticas tiene una política centralizada que es segmentada utilizando el atributo localidad y las extensiones de reglas produciendo políticas para las localidades. El sistema Kerberos es empleado para aprovechar las fases de autenticación y autorización utilizadas en el proceso de actualización de políticas entre el servidor y los clientes. El Apéndice C presenta un resumen del sistema Kerberos.

4.1. Ambientes Distribuidos

El diseño de políticas de SELinux implica determinar roles, dominios, tipos y niveles. La ingeniería de roles es utilizada para definir los roles necesarios para una organización, al igual que la jerarquía de roles representada por la dominancia entre roles. Los procesos de usuario y del sistema operativo son clasificados en dominios, en donde un rol puede tener acceso a uno o más dominios. Los dominios pueden tener transiciones a otros dominios para acceder a los objetos de un nivel de clasificación mayor o igual. En todo ambiente distribuido es necesario diseñar el conjunto de roles que podran ser ejercidos por los usuarios en las diferentes computadoras. Cabe mencionar que para SELinux se tiene que complementar el diseño de roles con el diseño de los atributos de dominio, tipo y nivel.

A continuación se presenta un ejemplo de un ambiente distribuido, para el cual solo se describen los roles, ya que son la parte de más alto nivel de abstracción a diferencia de los dominios, tipos, niveles, etc. El ejemplo del ambiente distribuido consiste en cinco computadoras con SELinux, en donde cada computadora es responsable de ofrecer un servicio de red. Para mostrar la asignación de roles se definen los usuarios *spike*, *jonh* y *bob*.

Roles para el ambiente distribuido

La Tabla 4.1 muestra el conjunto de roles que permiten realizar tareas administrativas (configuración, mantenimiento, etc.) relacionadas con los el sistema operativo y algunos servicios

de red.

Tabla 4.1: Roles

Rol	Descripción
bkpr	Rol para Backup
dbr	Admn. del servidor DB
dhcpr	Admn. del servidor DHCP
dnsr	Admn. del servidor DNS
msr	Admn. del servidor de Correo
systemr	Admn. del Sistema Operativo
wsr	Admn. del servidor Web

Roles y Computadoras de un ambiente distribuido

La Tabla 4.2 muestra los roles que se pueden ejercer en las diferentes computadoras del ambiente distribuido. Así por ejemplo el rol *systemr* puede ser ejercido en todas las computadoras del ambiente distribuido, ya que éste provee las funcionalidades para la configuración del sistema operativo. Obsérverse que existen varios roles que son comunes en varias computadoras y algunos roles que son solo parte del conjunto de roles de una computadora.

Tabla 4.2: Roles y Computadoras

	Computadora				
Rol	dbl	dhcpl	dnsl	msl	wsl
bkpr	x	x	x	x	x
dbr	x				
dhcpr		x			
dnsr			x		
msr				x	
systemr	x	x	x	x	x
wsr					x

Usuarios y Roles para una computadora

La Tabla 4.3 muestra los usuarios y roles para la computadora *dbl*. Para cada computadora se deben especificar los roles que pueden ejercer sus distintos usuarios. El usuario *spike* pueden ejercer en la computadora *dbl* los roles *bkpr, dbr* y *systemr*. Los usuarios *john* y *bob*

solo pueden ejercer los roles *systemr* y *bkpr*, respectivamente.

Tabla 4.3: Roles y Usuarios para la computadora dbl

	Usuarios		
Rol	spike	john	bob
bkpr	x		x
dbr	x		
dhcpr			
dnsr			
msr			
systemr	x	x	
wsr			

4.1.1. Compilación y actualización de políticas

Cada computadora con SELinux de un ambiente distribuido tiene asociada una política local. Esto implica que los administradores deben ejecutar de manera manual los procedimientos de carga, compilación y actualización de políticas en cada computadora que lo requiera. Consideramos que la política de una computadora es *actualizada* si integra los últimos cambios de las reglas. A continuación se citan dos ejemplos para el cambio de políticas.

Actualización de política para una computadora

Se require adicionar en la política de la computadora *dbl* las reglas necesarias para especificar que el usuario *spike* pueda ejercer el rol *wsr*. Esto implica una tarea sencilla, ya que el administrador solo necesita modificar, compilar y cargar la política de la computadora *dbl*.

Actualización de política para todas las computadoras

Se requiere modificar al rol *systemr* para que domine al rol *bkpr*. Esto implica adicionar reglas en las computadoras que permiten ejercer al rol *systemr*. Según la Tabla 4.2 el rol *systemr* puede ser ejercido en todas las computadoras del ambiente distribuido, por lo que todas las políticas del ambiente distribuido deberán ser modificadas, compiladas y cargadas.

La Figura 4.1 muestra los cambios necesarios para actualizar las políticas del ambiente distribuido (M-Modificar, A-Adicionar), de tal manera que la política de la computadora *dbl* requiere adicionar al rol *wsr* y modificar al rol *systemr*. El resto de las políticas solo requieren modificar al rol *systemr*, y posteriormente a los cambios es necesario compilar y cargar la política en cada computadora.

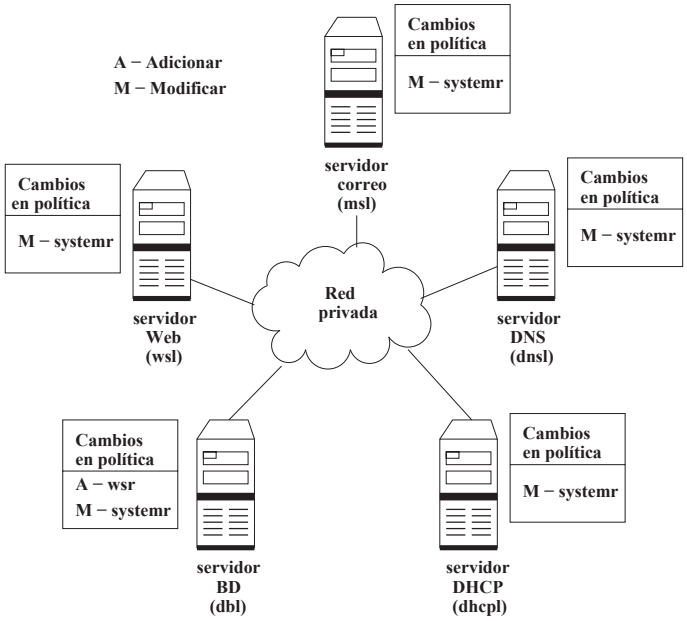

Figura 4.1: Actualización manual de políticas

Si un conjunto de reglas de la política cambia, y tales cambios son válidos para ciertas computadoras, el administrador de política debe actualizar la política en todas las computadoras que sean afectadas para obtener consistencia en las políticas. En la Figura 4.2 se muestra un ejemplo de la actualización manual de política, que no es una situación deseable ya que el administrador puede omitir el actualizar a ciertas computadoras; estas omisiones generan diferencias en las políticas, tal como sucede con las computadoras *wsl* y *dnsl*. Esto corresponde a un error de actualización manual de política, ya que por alguna razón el administrador omitió actualizar a todas la computadoras del ambiente distribuido.

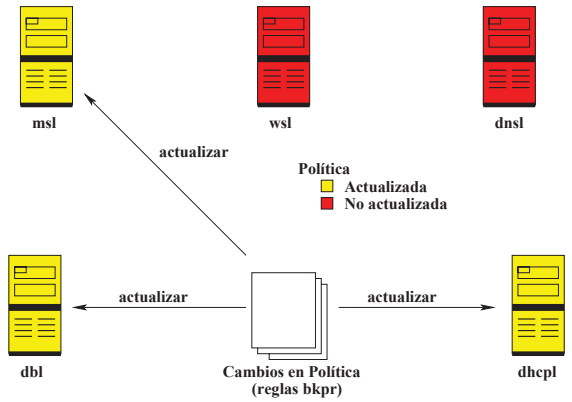

Figura 4.2: Error en la actualización manual

La Tabla 4.4 muestra el número de usuarios y roles de tres organizaciones [Kern02]. Como se observa para una organización grande se requiere administrar un gran número de roles, lo que implica una tarea administrativa que permita obtener confiabilidad en la operación de la organización [Zhang02].

Tabla 4.4: Usuarios y Roles

Tipo de Organización	Usuarios & Roles
Compañía Aseguradora	1700 usuarios 120 roles
Banco A	30000 usuarios (usuarios con promedio de 10 roles) 400 roles
Banco B	31000 usuarios 450 roles (en promedio 1 rol asignado a 70 usuarios)

La consistencia, autenticidad, integridad, disponibilidad y la confidencialidad son aspectos que deben presentarse en el proceso de actualización de política. Por estas razones es necesario desarrollar herramientas que sistematicen al proceso de administración de políticas. La complejidad de las políticas de SELinux y un gran número de roles son factores que se combinan para una administración compleja de las políticas de SELinux.

4.2. Atributo de localidad

Se define el concepto de *localidad* para denotar a cualquier computadora integrada en un ambiente distribuido. Para integrar este nuevo atributo en términos de las políticas de SELinux, se agregan dos nuevas reglas que permiten representar las relaciones entre usuarios, roles y localidades que se presentan en los ambientes distribuidos.

4.2.1. Reglas extendidas de SELinux

Regla para relación localidad-roles (l-r)

Se define una regla para las relaciones entre localidades y roles, la cual define al conjunto de roles que pueden ejercer los usuarios en cada localidad. Esta regla es una restricción que limita los roles para cada localidad.

Sintaxis: location l roles $\{r_1, r_2, ..., r_n\}$

donde l representa a una localidad y $r_1, r_2, ..., r_n$ representan los roles que pueden ejercerse en tal localidad.

Ejemplo: *location wsl roles {systemr, wsr, bkpr};*

Esta declaración define que los usuarios en la localidad *wsl* pueden ejercer el conjunto de roles *systemr, wsr, bkpr*.

Regla para relación usuario-localidad-roles (u-l-r)

Se define una regla para las relaciones entre usuario, localidad y roles, la cual define el conjunto de roles que un usuario puede ejercer en una localidad. El conjunto de roles especificado en esta regla debe ser igual o ser un subconjunto del conjunto de roles especificado por la regla para la relación localidad-roles.

Sintaxis: user u location l roles $\{r_1, r_2, ..., r_x\}$

donde u, l y $r_1, r_2, ..., r_x$ corresponde a un usuario, localidad y a un conjunto de roles, respectivamente.

Ejemplo: *user bob location wsl roles {bkpr};*

Esta declaración define que el usuario *bob* puede ejercer al rol *bkpr* en la localidad *wsl*.

4.2.2. Inclusión del atributo extendido de localidad

Para representar de una manera formal el atributo de localidad y las relaciones entre usuarios, localidad y roles se toma como base el modelo de control de acceso de SE-Linux presentado en el Capítulo 3.5.

Conjunto de localidades

Se define el conjunto L como el conjunto de localidades que integran el ambiente distribuido.

1. $L = \emptyset$
2. para cada regla del tipo $location(l, r_1, ..., r_n)$,
   - $L = L \cup l$

Conjunto de roles para una localidad

Cada localidad define a un conjunto válido de roles que pueden ejercer los usuarios. R_l es el conjunto de los roles válidos para la localidad l:

1. $R_l = \{object\_r\}$
2. para cada regla del tipo $location(l, r_1, ..., r_n)$,
   - con $i = 1, ..., n : R_l = R_{\ l} \cup r_i$
3. para cada $r \in R_l$ se obtiene al conjunto D(r) que corresonde al conjunto de roles de menor jerarquía del rol r,
   - $R_l = R_l \cup D(r)$

Conjunto de dominios para una localidad

T_l es el conjunto de todos los dominios autorizados para todos los roles que se pueden ejercer en la localidad l:

1. $T_l = \emptyset$

2. para cada regla del tipo $role(r, t_1, ..., t_n)$,

   - con $i = 1, ..., n$ y $r \in R_l$: $T_l = T_l \cup t_i$

Conjunto de usuarios para una localidad

U_l es el conjunto de usuarios para la localidad l:

1. $U_l = \emptyset$

2. para cada regla del tipo $user(u, l, r_1, ..., r_m)$,

   - con $l \in L$, $r_1, ..., r_m \in R_l$: $U_l = U_l \cup u$

4.3. Servidor de políticas

La Figura 4.3 muestra la nueva estructura de directorios para políticas locales, a la cual se le adiciona un directorio para representar las reglas para las relaciones entre localidad-roles y usuario-localidad-roles.

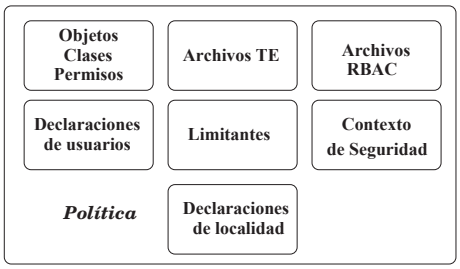

Figura 4.3: Estructura de Política para localidades

Una vez concentrada la política es necesario tener una herramienta que la segmente en políticas individuales para cada localidad del ambiente distribuido. Para esto se desarrolló una herramienta que obtiene políticas de localidades basándose en el análisis de las reglas para las relaciones localidad-roles y usuario-localidad-roles. En la Figura 4.4 se

muestra el esquema de segmentación de políticas, en donde la política es segmentada utilizando el atributo localidad y las extensiones de reglas correspondientes a las relaciones localidad-roles y usuario-localidad-roles.

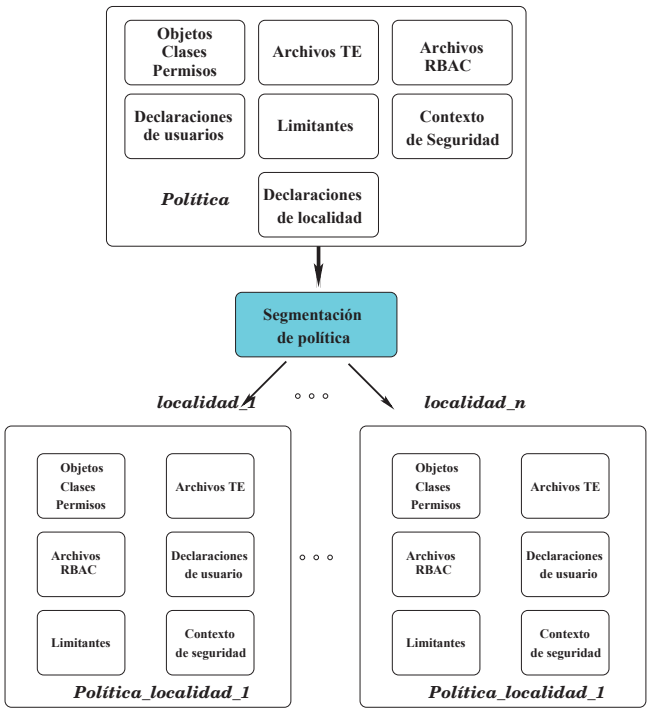

Figura 4.4: Segmentación de Políticas

En el ambiente distribuido cada localidad tiene asociada una política SELinux; las políticas de las localidades pueden ser iguales, semejantes o diferentes. Las políticas están formadas por las reglas de los modelos de control de acceso (dominios, tipos, roles y niveles), reglas de especificación de usuarios, limitantes, etc.

La Figura 4.5 muestra la arquitectura desarrollada, de la cual una localidad es designada para operar como servidor de políticas, con la finalidad de concentrar, asignar y actualizar políticas. Los atributos de confidencialidad, integridad y autenticidad deben estar presentes en los procesos de concentración, asignación y actualización de políticas. El sistema Kerberos, instalado en otra localidad, es empleado para lograr obtener los atributos confidencialidad, integridad y autenticidad, necesarios en el proceso administrativo

de políticas. Para obtener la disponibilidad de políticas proponemos agregar un servidor de políticas redundante, el cual no es desarrollado en este trabajo y se propone como un trabajo futuro.

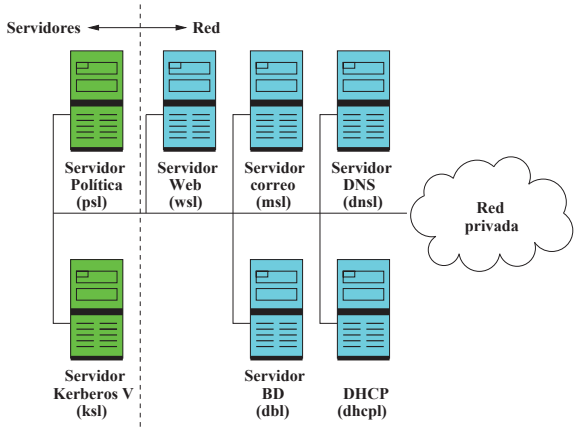

Figura 4.5: Arquitectura propuesta para la administración de políticas

El centralizar políticas evita generar inconsistencias en el ambiente distribuido cuando las reglas cambian. El servidor de políticas asigna y actualiza las políticas de las localidades que necesiten ser actualizadas.

4.3.1. Proceso de asignación de política

Cuando una localidad se integra al ambiente distribuido, necesita contar con una identidad dentro del ambiente. Algunas localidades tendrán la configuración estática de los parámetros de red y otras tendrán que obtenerlos de manera dinámica. En la Figura 4.5 se tiene un servidor responsable del servicio dhcp para asignar los siguientes parámetros necesarios en la operación distribuida:

- Parámetros de red: ip, netmask y servidores para dns, dhcp, ntp.
- Servidor de Kerberos: Nombre del reyno, servidores maestro y esclavo.
- Servidor de Políticas: Servidores maestro y esclavo.

Una vez que una localidad tiene (estáticamente ó dinámicamente) los parámetros de red, procede a interactuar con el sistema Kerberos en las fases de autenticación y autorización. Para la fase de servicio, el cliente le envía la clave de sesión $K_{C,S}$ al servidor de política. La clave de sesión debe proporcionar integridad y confidencialidad en el proceso de actualización de política. Una vez que la localidad comparte una clave de sesión con el servidor de políticas, la localidad le envía el valor md5 correspondiente a su política, tal como se muestra en el mensaje 1 del Protocolo 1 (protocolo para la verificación de modificación). Si los valores md5 de la localidad y el servidor son iguales, significa que las políticas no han cambiado. En caso de valores md5 diferentes la política de la localidad tiene que ser actualizada.

Protocolo 1: Verificación de modificación.

C-Cliente, y *S-Servidor* de política.

1. $C \rightarrow S : \{md5(policy)\}_{K_{C,S}}$
   - IF valores md5 son iguales THEN
2. $S \rightarrow C : \{Ok\}_{K_{C,S}}$
   - ELSE
3. $S \rightarrow C : \{Update\}_{K_{C,S}}$
4. $S \rightarrow C : \{Newpolicy\}_{K_{C,S}}$
5. $C \rightarrow S : \{Ok\}_{K_{C,S}}$

Descripción del protocolo para verificación de modificación

El cliente de política le envía al servidor de política el mensaje 1, el cual contiene el valor md5 de su política. El servidor compara el valor md5 recibido y el valor md5 correspondiente a la política del cliente contenida en el servidor. Si los valores md5 son iguales entonces el servidor de políticas le envía al cliente el mensaje 2, con el cual se le indica al cliente que su política está actualizada. En el caso de valores md5 distintos, entonces el servidor de políticas le envía al cliente el mensaje 3, indicándole al cliente que su política debe ser actualizada. Con el mensaje 4 el servidor le envía la política al cliente. Una vez que el cliente ha recibido la política, éste la actualiza, compila y carga, y posteriormente le envía al servidor de políticas el mensaje 5 donde le indica que la política fue actualizada.

4.3.2. Proceso de actualización de política

El ejemplo citado en la Sección 4.1.1 es solucionado mediante la arquitectura propuesta. Para este ejemplo la Figura 4.6 se muestra que las políticas segmentadas indican que se requiere actualizar solo la política de la localidad *dbl*, debido a la adición del rol *wsr* en las reglas de la política.

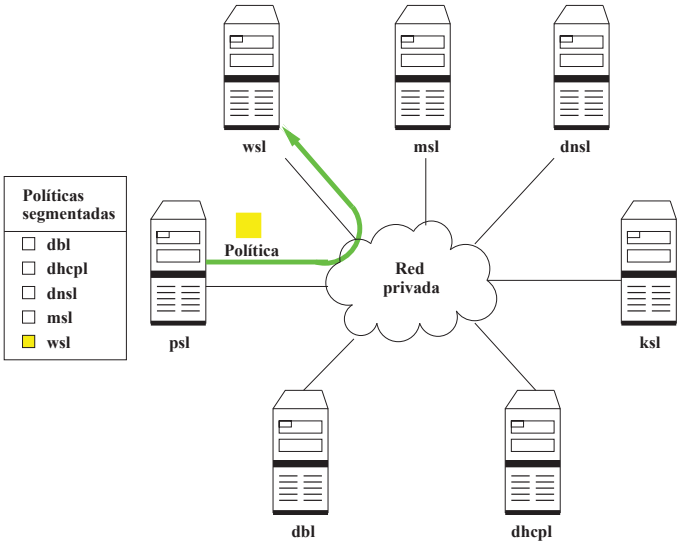

Figura 4.6: Actualización de política en una localidad

La Figura 4.7 muestra la actualización de todas las políticas del ambiente distribuido, ya que el rol *systemr* es modificado para dominar al rol *bkpr*. Esto es debido a que el rol *systemr* puede ser ejercido en todas la localidades del ambiente distribuido.

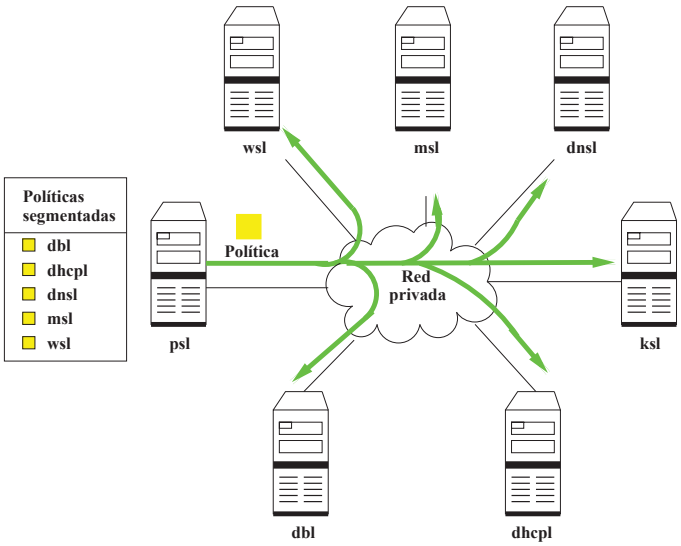

Figura 4.7: Actualización de política en todas las localidades

El ejemplo anterior ilustra de una manera breve, como la arquitectura propuesta elimina la necesidad de actualizaciones manuales en cada locaidad, facilitando el trabajo de administración y asegurando que las políticas en cada localidad están siempre actualizadas con respecto al servidor de políticas.

4.4. Conclusiones

En este capítulo se describió la arquitectura propuesta para sistemas SELinux empleados en ambientes distribuidos. Tal arquitectura tiene la finalidad ofrecer para los ambientes distribuidos una administración de políticas SELinux sencilla y confiable, eliminando los problemas de inconsistencias e inseguridad. Se describieron los problemas que se presentan en los ambientes distribuidos que emplean computadoras con SELinux; se describe al atributo localidad y las reglas extendidas de SELinux que consideran a tal atributo. Se describieron los procesos de asignación y actualización de políticas.

Capítulo 5

Resultados Obtenidos

Un servidor de políticas y tres clientes de políticas *ws_l*, *amd64*, *ms_l*, fueron utilizados para mostrar los siguientes resultados. El archivo de configuración que especifica la relaciones localidad-roles (l-r) y usuario-localidad-roles (u-l-r) se compone por trece reglas de las cuales las reglas 1,2,3 y 4 se aplican a la localidad *ws_l*, las reglas 5,6,7,8 y 9 se aplican a la localidad *amd64* y las reglas 10,11,12 y 13 se aplican a la localidad *ms_l*.

Archivo de relaciones l-r y u-l-r

```
Linea
1.  location ws_l roles { user_r sysadm_r system_r };
2. user system_u location ws_l roles system_r;
3.  user user_u location ws_l roles { user_r sysadm_r system_r };
4. user root location ws_l roles { user_r system_r };

5. location amd64 roles { user_r sysadm_r system_r };
6. user system_u location amd64 roles system_r;
7. user user_u location amd64 roles { user_r sysadm_r system_r };
8. user root location amd64 roles { user_r sysadm_r system_r };
9. user pedro location amd64 roles { user_r sysadm_r system_r };

10. location ms_l roles { sysadm_r system_r };
11. user system_u location ms_l roles system_r;
12. user user_u location ms_l roles { user_r sysadm_r system_r };
13. user root location ms_l roles { sysadm_r system_r };
```

5.1. Cliente y Servidor

Un cliente de política al momento de ser encendido o restablecido ejecuta los procesos de comparación de políticas y, solo en caso de ser necesario, el proceso de actualización, compilación y carga de política. Por otra parte el servidor de políticas ejecuta el proceso

de actualización de políticas en las localidades cuando en el proceso de segmentación de políticas se tienen valores md5 diferentes entre las políticas segmentadas y las políticas actuales.

Lado cliente

La Figura 5.1 muestra una terminal con SELinux durante el proceso de verificación y actualización de política parte de la localidad *amd64*. En esta pantalla se muestran distintos valores de md5 correspondientes a la política segmentada para la localidad *amd64* y la política actual de esta localidad contenida en el servidor. Al existir diferencia entre los valores md5 correspondiente a la localidad *amd64* entonces se procede a ejecutar los procesos de actualización, compilación y cargar la política de tal localidad.

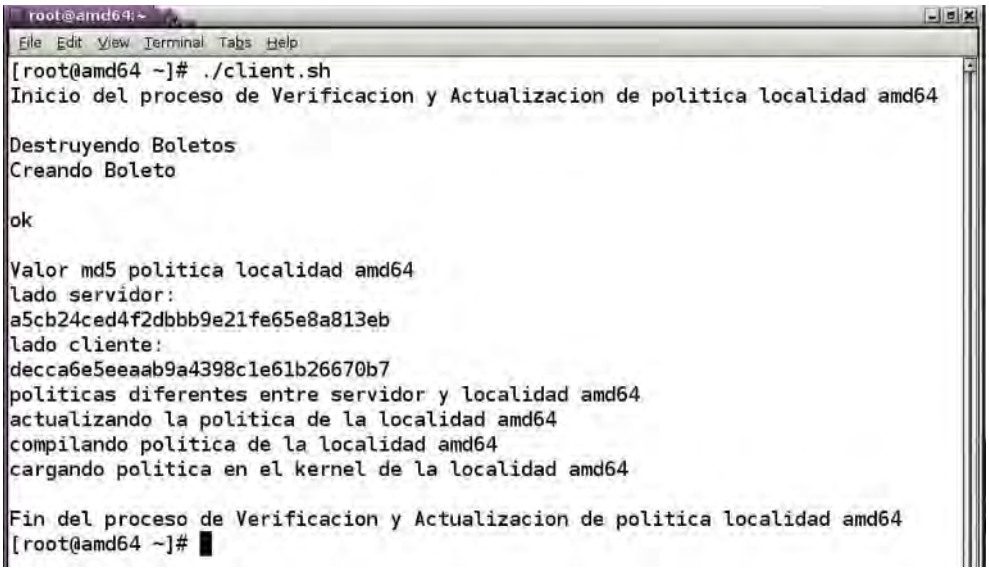

Figura 5.1: Lado cliente caso no actualizado

La Figura 5.2 muestra también el proceso de verificación y actualización de política para la localidad *amd64*, se muestran iguales valores de md5 entre la política de la localidad amd64 y la política correspondiente a esta localidad contenida en el servidor, por lo que no es necesario actualizar, compilar y cargar la política en la localidad amd64.

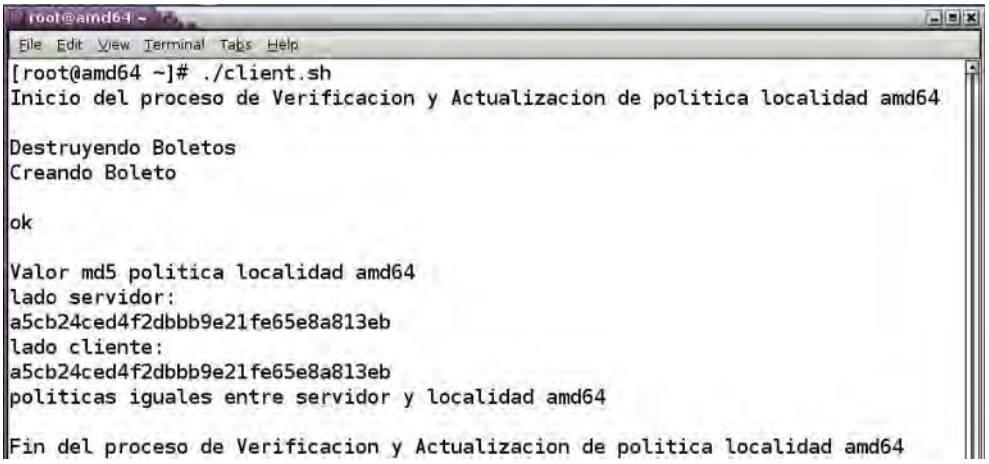

Figura 5.2: Lado cliente caso actualizado

Como se ha mostrado en las Figuras 5.1 y 5.2 la administración de las políticas SELinux es automatizada con el fin de evitar errores por parte del administrador de políticas. Aunque estas figuras muestran que los programas están siendo ejecutados manualmente, estas instrucciones pueden ser incluidas en los scripts que se ejecutan cuando la computadora se inicia.

Lado servidor

La Figura 5.3 muestra el proceso de verificación y actualización de políticas por parte del servidor de políticas, se observan distintos valores de md5 entre la política de la localidad amd64 y la política correspondiente a esta localidad contenida en el servidor. Por consiguiente se actualiza, compila y carga la política en la localidad amd64.

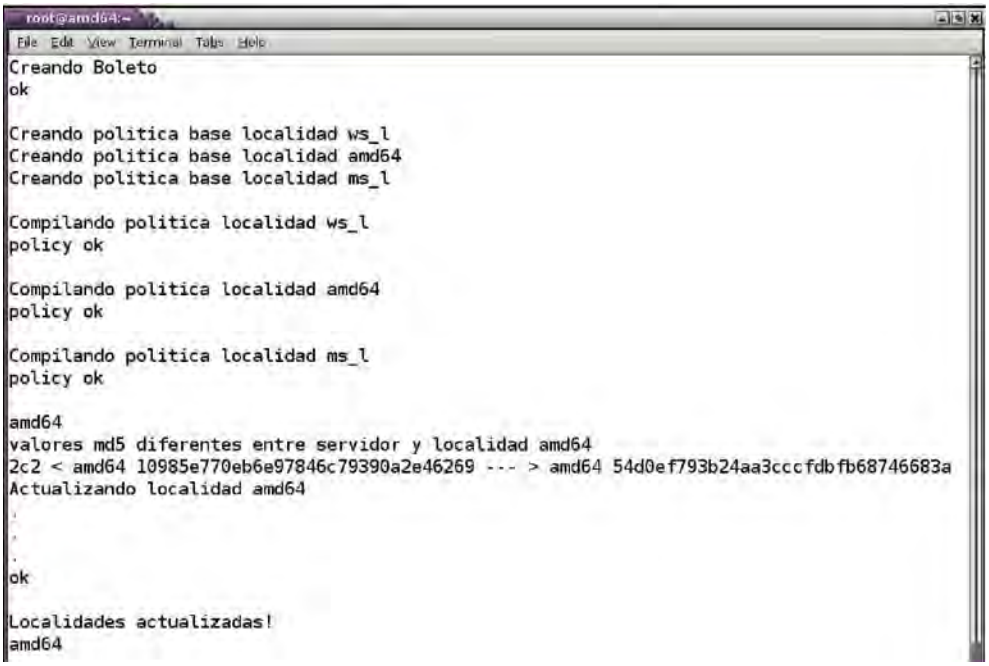

Figura 5.3: Lado servidor caso no actualizado

La Figura 5.4 muestra también el proceso de verificación y actualización de políticas por parte del servidor de políticas. En este caso se tienen valores iguales de md5 entre las políticas de las localidades y la política correspondiente a cada localidad contenida en el servidor. Por consiguiente no es necesario actualizar, compilar y ni cargar la política de alguna localidad.

Figura 5.4: Lado servidor caso actualizado

El servidor de políticas es el encargado de segmentar la política centralizada y producir políticas locales para los clientes. Además de ejecutar el procedimiento de actualización en los clientes que lo requieran debido a cambios en sus políticas.

5.2. Negación y Otorgamiento de rol

Para mostrar la negación y el otorgamiento de roles se modifica las relaciones *l-r* y *u-l-r* mostradas al inicio de esta sección. Los cambios se realizan sobre la localidad *amd64*, el usuario *pedro* y el rol *sysadm_r*; específicamente se modifica la línea número 9 del archivo de relaciones l-r y u-l-r mostradas al inicio de esta sección.

El comando *newrole* permite que los usuarios tengan acceso a un rol específico, para lo cual es necesario especificar, mediante la opción *-r*, el rol objetivo. Mediante el comando *id* y la opción *-Z* se puede conocer el contexto de seguridad asociado a un usuario.

Negación de rol

La Figura 5.5 muestra la negación del rol *sysadm_r* para el usuario *pedro* en la localidad *amd64*. Esto se cumple debido a que no existe de una regla que le permita a tal usuario tener acceso a dicho rol, por lo tanto el contexto de seguridad solicitado es inválido (user

pedro location amd64 roles {user_r, system_r};).

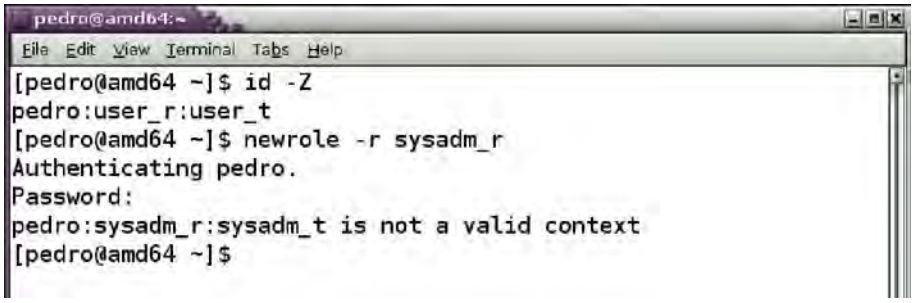

Figura 5.5: Negación de rol

Otorgamiento de rol

La Figura 5.6 muestra el otorgamiento del rol *sysadm_r* para el usuario *pedro* en la localidad *amd64*. Esto se cumple debido a la existencia de una regla que le permita a tal usuario tener acceso a dicho rol, por lo tanto el contexto de seguridad solicitado es válido (user pedro location amd64 roles {user_r, sysadm_r, system_r};).

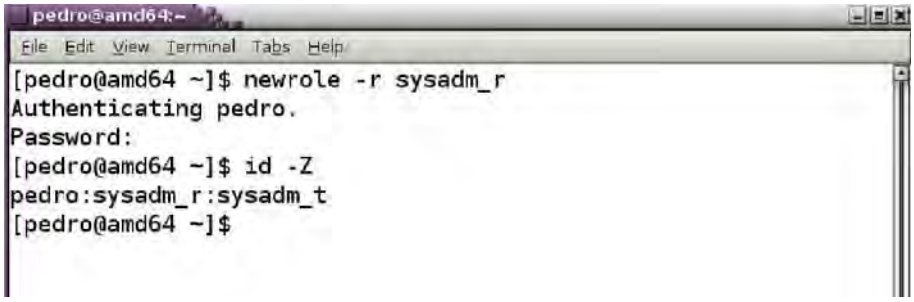

Figura 5.6: Otorgamiento de rol

5.3. Conclusiones

Para el desarrollo de este trabajo se propuso un conjunto de reglas que permiten la integración de SELinux en ambientes distribuidos. Para analizar tal conjunto de reglas se desarrolló una herramienta que segmenta y obtiene las diferentes políticas del entorno distribuido. Tal herramienta está implementada en el lenguaje script bash y se compone de

dos partes; una parte para el servidor de políticas y la otra para el cliente. Los pasos para la implementación de Kerberos, arranque del cliente y servidor dependen de cada distribución de Linux, por lo que no son parte de este trabajo.

La biblioteca *lkrb5* se utilizó para implementar funciones en C de las fases de autenticación, autorización y servicio. Por desgracia esta biblioteca no se encuentra bien documentada y además se tienen dos implementaciones diferentes; MIT [mit09] y Heimdal [hei09]. Algunos desarrolladores de kerberos del MIT recomendaron utilizar la biblioteca GSSAPI en lugar de la biblioteca de kerberos del MIT. Dada la inadecuada documentación de las bibliotecas, se decidió utilizar servicios kerberizados mediante la programación del bash.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

En este trabajo abordamos los problemas que se presentan en los ambientes distribuidos que integran a sistemas SELinux y proporcionamos una arquitectura que los resuelve de una manera sencilla y confiable. Integramos un conjunto de reglas para SELinux que permiten especificar roles y usuarios para las diferentes localidades del ambiente distribuido. Hacemos uso del sistema Kerberos para obtener integridad, confidencialidad y autenticidad en el proceso de actualización de políticas.

En la sección 4.2.2 se formalizó la extensión del atributo localidad. Con la presente solución se eliminan los problemas de falta de actualización de políticas e inseguridad en la actualización presentados en los ambientes distribuidos.

En este trabajo se ofrece también un marco de referencia del control de acceso, para lo cual se presentan el origen, los tipos, algunos modelos y el control de acceso de SELinux.

Otro de los problemas presentes en los ambientes distribuidos de sistemas SELinux es el distinguir entre sesiones de usuario local o remoto, este problema se cita en la sección 6.2. Para el proceso de actualización de políticas surge la pregunta relacionada con las activida-

des de los usuarios; ya que se pueden agregar reglas que impidan realizar las actividades de un usuario que se encuentra en línea. La co-dependencia entre políticas remotas y locales es otro aspecto importante a resolver en los sistemas distribuidos, este problema es presentado en la sección 6.3.

Para el ambiente distribuido se obtiene la necesidad de definir localidades de confianza distinguiendo entre sesiones locales y remotas, además de la necesidad de integrar al control de acceso tradicional condiciones de mal uso de recursos, reservación de recursos y marcas de tiempo, problema se citado en la sección 6.4.

Las investigaciones del presente trabajo pudieron ser publicadas en cinco diferentes foros internacionales y un foro nacional [Ch.L.Pedro06, Ch.L.Pedro07, Ch.L.Pedro08b, Ch.L.Pedro08c, Ch.L.Pedro08a].

6.2. Sesiones Locales y Remotas

La idea básica consiste en distinguir entre las sesiones locales y remotas para definir usuarios y localidades de confianza. La integración de localidades de confianza permite definir políticas locales y remotas. Los modelos actuales y las implementaciones para el control de acceso son deficientes en este sentido ya que no consideran la distinción de sesiones locales y remotas. Un ejemplo de esto se presenta en la Tabla 6.1, la cual muestra los diferentes roles que un usuario puede ejercer para sesiones locales y remotas.

Tabla 6.1: Roles para el usuario spike incluyendo atributos Desde y Para

Desde	Para dbl	dhcpl	dnsl	msl	wsl
dbl	bkpr dbr systemr	bkpr	bkpr	bkpr	bkpr wsr
dhcpl	bkpr	bkpr	bkpr	bkpr	bkpr
dnsl	bkpr	bkpr	bkpr	bkpr	bkpr
msl	bkpr	bkpr	bkpr	bkpr	bkpr
wsl	bkpr dbr	bkpr	bkpr	bkpr	systemr wsr

Para la distinción entre sesiones locales y remotas es necesario adicionar el atributo localidad al contexto de seguridad de SELinux, tal como se muestra en la Figura 6.1. Esto implica modificaciones considerables a las estructuras de SELinux, compilador, macros, etc.

Scontext	**user_u**	**webmaster_r**	**webmaster_t**	**s0:c0.c2**	**webserver_l**
Atributo	Identidad	rol	tipo	nivel	localidad
Modelo	User Identity	RBAC	TE	MLS	Localidad

Figura 6.1: Nuevo Contexto de Seguridad

El acceso remoto a computadoras o servidores de una organización es utilizado por los empleados de una organización para accesos realizados desde lugares externos a la organización.

el cual se realiza desde un lugar externo a la organización. La comunicación entre el equipo utilizado por el empleado y el servidor es cifrada utilizando una red privada virtual (VPN); la información puede ser interceptada pero no podrá ser interpretada instantáneamente. La pregunta obligada es, Qué pasa con la información en la parte del empleado? ya que el empleado puede ser engañado, amenazado, obligado o por iniciativa propia de sustraer información, etc. Una solución alterna a este problema es bloquear a los accesos remotos mediante reglas del Firewall. Pero puede existir la necesidad de intercambiar infor-

mación no tan importante entre organización y empleados lo cual no es posible controlar con las reglas de Firewall. En este escenario se necesita que el control de acceso limite el acceso remoto de los recursos de información.

6.3. Co-dependencia entre políticas

La co-dependencia entre políticas es otro aspecto importante a considerar en las sesiones locales y remotas. La co-dependencia entre políticas se refiere a la relación que existe entre diferentes políticas, lo cual implica dependencia ente las políticas locales y remotas del ambiente distribuido para realizar las diferentes actividades de los usuarios. Para la Figura 6.2 un usuario inicia una sesión local en la localidad *wsl* y ejerce el rol *wsr*. Ciertas tareas del rol *wsr* requieren acceder a las bases de datos de la localidad *dbl*, para lo cual es necesario ejercer al rol *dbr*. Esta operación genera un problema de acceso ya que en la localidad *dbl* el usuario no puede ejercer el rol *dbr* desde la localidad *wsl*, lo cual es una contradicción para el rol *wsr*.

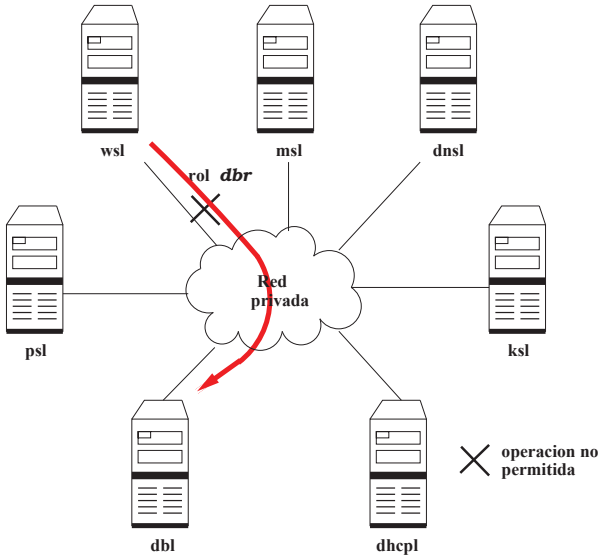

Figura 6.2: Co-dependencia entre políticas

6.4. Mal uso de Recursos

A pesar de que se tienen grandes avances en el desarrollo de SELinux, no se tiene una proyección que considere al mal uso de recursos y no existe un modelo o sistema de control de acceso que especifique marcas de tiempo, cantidad de acceso permitidos y la reservación de recursos.

La Figura 6.3, muestra la idea fundamental de restringir el número de accesos permitidos a un objeto o un conjunto de objetos. Lo cual eliminará la falla del sistema por gusanos o virus que intenten reproducirse o ejecutarse un cierto número de veces.

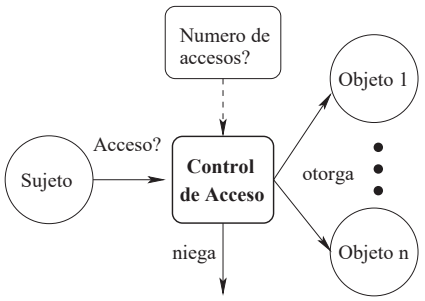

Figura 6.3: Número de Accesos

La Figura 6.4, muestra la restricción del control de acceso mediante marcas de tiempo, lo cual permitirá especificar fechas y horarios laborables de la organización.

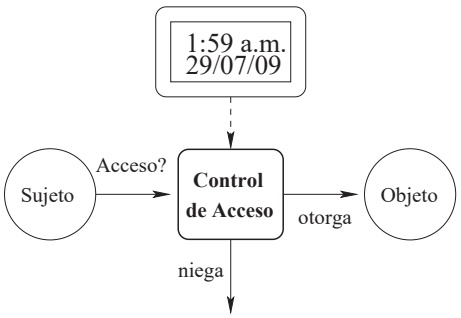

Figura 6.4: Marca de Tiempo para el Control de Acceso

La especificación por mal uso de recursos debe manejarse mediante la reservación de recursos, ya que un usuario puede necesitar acceso a más recursos o ejecutar a un mayor número de procesos que otro usuario.

6.5. Trabajos Futuros

Desarrollar e integrar a la arquitectura propuesta un servidor redundante de políticas, considerando tolerancias a fallas. Para esto se deben plantear diversos escenarios de fallas que se presentan en los ambientes distribuidos, tal como la pérdida de comunicación, etc.

Analizar y diseñar una herramienta que determine la co-dependencia para diferentes políticas, de tal manera que las políticas en un ambiente distribuido sean consistentes desde el punto de vista de la dependencia entre roles, dominios y tipos.

Analizar posibles soluciones para la actualización de políticas considerando el caso de actualización en condiciones de operación por parte de los usuarios. Una posible alternativa de solución se puede encontrar empleando al protocolo de asignación de dos fases (Two-phase commit protocol).

Para los ambientes distribuidos diseñar y desarrollar un servidor de políticas que administre políticas heterogéneas, de tal manera que se puedan integrar las políticas de seguridad de

los sistemas operativos Windows, SELinux, Solaris, etc.

Diseñar e implementar para los sistemas SELinux reglas que permitan controlar el mal uso de recursos. Este problema es clásico en la gran mayoría de las implementaciones de control de acceso y es debido a la naturaleza de los modelos de control de acceso empleados.

Proponer un nuevo modelo de control de acceso que integre el concepto de localidad de confianza y proporcione un manejo adecuado de los recursos mediante accesos remotos. De tal manera que se tenga políticas que se ejecuten cuando un usuario accede desde una localidad remota. Además analizar el atributo de localidad como un dominio correspondiente a un conjunto de equipos.

Proponer una nueva definición de los tipos de control de acceso, ya que actualmente se maneja una definición basada en los ambientes militares de los años sesentas.

Apéndice A

Estructuras Básicas de SELinux

```
typedef struct policy_file {
#define PF_USE_MEMORY  0
#define PF_USE_STDIO   1
#define PF_LEN         2 /* total up length in len field */
unsigned type;
char *data;
size_t len;
size_t size;
FILE *fp;
struct sepol_handle *handle;
unsigned char buffer[BUFSIZ];
} policy_file_t;

/* The policy database */
typedef struct policydb {
#define POLICY_KERN SEPOL_POLICY_KERN
#define POLICY_BASE SEPOL_POLICY_BASE
#define POLICY_MOD SEPOL_POLICY_MOD
uint32_t policy_type;
char *name;
char *version;

/* Set when the policydb is modified such that writing is unsupported */
int unsupported_format;

/* Whether this policydb is mls, should always be set */
int mls;

/* symbol tables */
symtab_t symtab[SYM_NUM];
#define p_commons symtab[SYM_COMMONS]
#define p_classes symtab[SYM_CLASSES]
#define p_roles symtab[SYM_ROLES]
#define p_types symtab[SYM_TYPES]
#define p_users symtab[SYM_USERS]
#define p_bools symtab[SYM_BOOLS]
#define p_levels symtab[SYM_LEVELS]
#define p_cats symtab[SYM_CATS]

/* symbol names indexed by (value - 1) */
char **sym_val_to_name[SYM_NUM];
#define p_common_val_to_name sym_val_to_name[SYM_COMMONS]
```

```
#define p_class_val_to_name sym_val_to_name[SYM_CLASSES]
#define p_role_val_to_name sym_val_to_name[SYM_ROLES]
#define p_type_val_to_name sym_val_to_name[SYM_TYPES]
#define p_user_val_to_name sym_val_to_name[SYM_USERS]
#define p_bool_val_to_name sym_val_to_name[SYM_BOOLS]
#define p_sens_val_to_name sym_val_to_name[SYM_LEVELS]
#define p_cat_val_to_name sym_val_to_name[SYM_CATS]

/* class, role, and user attributes indexed by (value - 1) */
class_datum_t **class_val_to_struct;
role_datum_t **role_val_to_struct;
user_datum_t **user_val_to_struct;
type_datum_t **type_val_to_struct;

/* module stuff section -- used in parsing and for modules */

/* keep track of the scope for every identifier.  these are
 * hash tables, where the key is the identifier name and value
 * a scope_datum_t.  as a convenience, one may use the
 * p_*_macros (cf. struct scope_index_t declaration). */
symtab_t scope[SYM_NUM];

/* module rule storage */
avrule_block_t *global;
/* avrule_decl index used for link/expand */
avrule_decl_t **decl_val_to_struct;

/* compiled storage of rules - use for the kernel policy */

/* type enforcement access vectors and transitions */
avtab_t te_avtab;

/* bools indexed by (value - 1) */
cond_bool_datum_t **bool_val_to_struct;
/* type enforcement conditional access vectors and transitions */
avtab_t te_cond_avtab;
/* linked list indexing te_cond_avtab by conditional */
cond_list_t *cond_list;

/* role transitions */
role_trans_t *role_tr;

/* role allows */
role_allow_t *role_allow;

/* security contexts of initial SIDs, unlabeled file systems,
   TCP or UDP port numbers, network interfaces and nodes */
ocontext_t *ocontexts[OCON_NUM];

/* security contexts for files in filesystems that cannot support
   a persistent label mapping or use another
   fixed labeling behavior. */
genfs_t *genfs;

/* range transitions */
range_trans_t *range_tr;

ebitmap_t *type_attr_map;

ebitmap_t *attr_type_map; /* not saved in the binary policy */

unsigned policyvers;
} policydb_t;
```

Apéndice B

Modelo formal del Control de Acceso de SELinux

En este apéndice se presentan los detalles de los conjuntos del modelo de control de acceso de SELinux presentado en el Capítulo 3.5.

El conjunto C tiene como elementos las clases definidas por la arquitectura *Flask* :
$C = \{security, process, system, capability, file, filesystem, dir,$
$fd, lnk\_file, chr\_file, blk\_file, sock\_file,$
$fifo\_file, socket, tcp\_socket, udp\_socket,$
$rawip\_socket, node, netif, netlink\_socket, key\_socket, packet\_socket,$
$unix\_stream\_socket, unix\_dgram\_socket, sem, msg, msgq, shm, ipc, passwd, drawable,$
$window, gc, font, colormap, property, cursor, xclient, xinput, xserver, xextension, pax,$
$netlink\_route\_socket, netlink\_firewall\_socket, netlink\_tcpdiag\_socket,$
$netlink\_nflog\_socket, netlink\_xfrm\_socket, netlink\_selinux\_socket,$
$netlink\_audit\_socket, netlink\_ip6fw\_socket, netlink\_dnrt\_socket, dbus, nscd,$
$association, netlink\_kobject\_uevent\_socket\}$

Existen permisos comunes para la clase socket, file, ipc:

Permisos comunes para socket

$P_{Socket} = \{append, bind, connect, create, write, relabelfrom,$
$ioctl, name\_bind, sendto, recv\_msg,$
$send\_msg, getattr, setattr, accept, getopt,$
$read, setopt, shutdown, recvfrom, lock, relabelto, listen\}$

Permisos comunes para archivos

$P_{File} = \{append, create, execute, write, relabelfrom,$
$link, unlink, ioctl, getattr, setattr, read,$
$rename, lock, relabelto, mounton, quotaon, swapon\}$

Permisos comunes para ipc

$P_{Ipc} = \{associate, create, write, unix\_read, destroy, getattr,$
$setattr, read, unix\_write\}$

Permisos para cada clase de objetos del conjunto C.

$P(security) = \{compute\_member, compute\_user, compute\_create,$
$setenforce, check\_context, setcheckreqprot,$
$compute\_relabel, setbool, load\_policy,$
$setsecparam, compute\_av\}$

$P(process) = \{getcap, setcap, sigstop, sigchld, share,$
$execheap, setcurrent, setfscreate, siginh,$
$dyntransition, transition, fork, getsession,$
$noatsecure, sigkill, signull, setrlimit,$
$getattr, getsched, setexec, setsched,$
$getpgid, setpgid, ptrace, execstack,$
$rlimitinh, signal, execmem\}$

$P(system) = \{ipc\_info, syslog\_read, syslog\_console, syslog\_mod\}$

$P(capability) = \{setpcap, fowner, sys\_boot, sys\_tty\_config, net\_raw,$
$sys\_admin, sys\_chroot, sys\_module, sys\_rawio, dac\_override, ipc\_owner, kill,$
$dac\_read\_search, sys\_pacct, net\_broadcast, net\_bind\_service, sys\_nice,$
$sys\_time, fsetid, mknod, setgid, setuid, lease, net\_admin, audit\_write,$
$linux\_immutable, sys\_ptrace, audit\_control, ipc\_lock, sys\_resource, chown\}$

$P(filesystem) = \{associate, quotaget, relabelfrom, transition,$
$getattr, quotamod, mount, remount, unmount, relabelto\}$

- $P(file) = \{entrypoint, execmod, execute\_no\_trans\}$
- $P(file) = P(file) \cup P_{File}$

- $P(dir) = \{rmdir, remove\_name, add\_name, reparent, search\}$
- $P(dir) = P(dir) \cup P_{File}$

- $P(fd) = \{use\}$
- $P(fd) = P(fd) \cup P_{File}$

- $P(lnk\_file) = \{\emptyset\}$
- $P(lnk\_file) = P(lnk\_file) \cup P_{File}$

- $P(chr\_file) = \{entrypoint, execmod, execute\_no\_trans\}$
- $P(chr\_file) = P(chr\_file) \cup P_{File}$

- $P(blk\_file) = \{\emptyset\}$
- $P(blk\_file) = P(blk\_file) \cup P_{File}$

- $P(fifo\_file) = \{\emptyset\}$
- $P(fifo\_file) = P(fifo\_file) \cup P_{File}$

- $P(sock\_file) = \{\emptyset\}$
- $P(sock\_file) = P(sock\_file) \cup P_{Socket}$

- $P(socket) = \{\emptyset\}$
- $P(socket) = P(socket) \cup P_{Socket}$

- $P(tcp\_socket) = \{acceptfrom, connectto, node\_bind, newconn, name\_connect\}$
- $P(tcp\_socket) = P(tcp\_socket) \cup P_{Socket}$

- $P(udp\_socket) = \{node\_bind\}$
- $P(udp\_socket) = P(udp\_socket) \cup P_{Socket}$

- $P(rawip\_socket) = \{node\_bind\}$
- $P(rawip\_socket) = P(udp\_socket) \cup P_{Socket}$

- $P(node) = \{rawip\_recv, tcp\_recv, udp\_recv, rawip\_send, tcp\_send, udp\_send, enforce\_dest\}$

- $P(netif) = \{rawip\_recv, tcp\_recv, udp\_recv, rawip\_send, tcp\_send, udp\_send\}$

- $P(netlink\_socket) = \{\emptyset\}$
- $P(netlink\_socket) = P(netlink\_socket) \cup P_{Socket}$

- $P(packet\_socket) = \{\emptyset\}$
- $P(packet\_socket) = P(netlink\_socket) \cup P_{Socket}$

- $P(key\_socket) = \{\emptyset\}$
- $P(key\_socket) = P(key\_socket) \cup P_{Socket}$

- $P(unix\_stream\_socket) = \{acceptfrom, connectto, newconn\}$
- $P(unix\_stream\_socket) = P(unix\_stream\_socket) \cup P_{Socket}$

- $P(unix\_dgram\_socket) = \{\emptyset\}$
- $P(unix\_dgram\_socket) = P(unix\_dgram\_socket) \cup P_{Socket}$

- $P(sem) = \{\emptyset\}$
- $P(sem) = P(sem) \cup P_{Ipc}$

- $P(msg) = \{send, receive\}$

- $P(msgq) = \{enqueue\}$

- $P(msgq) = P(msgq) \cup P_{Ipc}$

- $P(shm) = \{lock\}$
- $P(shm) = P(shm) \cup P_{Ipc}$

- $P(ipc) = \{enqueue\}$
- $P(ipc) = P(ipc) \cup P_{Ipc}$

$P(passwd) = \{chfn, crontab, passwd, chsh, rootok\}$
$P(window) = \{chprop, map, unmap, extensionevent, addchild,$
$windowchangeevent, ctrllife, setfocus, create, move, clientcomevent,$
$enumerate, listprop, serverchangeevent, destroy, chselection, inputevent,$
$chparent, getattr, setattr, transparent, drawevent, chproplist, chstack,$
$mousemotion, windowchangerequest\}$
$P(gc) = \{create, getattr, setattr, free\}$
$P(font) = \{getattr, load, free, use\}$
$P(colormap) = \{list, create, store, install, getattr, setattr, read, free, uninstall\}$
$P(property) = \{create, write, read, free\}$
$P(cursor) = \{create, createglyph, setattr, free, assign\}$
$P(xclient) = \{kill\}$
$P(xinput) = \{lookup, relabelinput, setfocus, activegrab, warppointer,$
$bell, getattr, setattr, ungrab, passivegrab, mousemotion\}$
$P(xserver) = \{gethostlist, screensaver, sethostlist, getfontpath, setfontpath, getattr, grab, ungrab\}$
$P(xextension) = \{use, query\}$
$P(pax) = \{mprotect, emutramp, randmmap, pageexec, randexec, segmexec\}$

- $P(netlink\_route\_socket) = \{nlmsg\_write, nlmsg\_read\}$

- $P(netlink\_route\_socket) = P(netlink\_route\_socket) \cup P_{Socket}$

- $P(netlink\_firewall\_socket) = \{nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_firewall\_socket) = P(netlink\_firewall\_socket) \cup P_{Socket}$

- $P(netlink\_tcpdiag\_socket) = \{nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_tcpdiag\_socket) = P(netlink\_tcpdiag\_socket) \cup P_{Socket}$

- $P(netlink\_nflog\_socket) = \{nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_nflog\_socket) = P(netlink\_nflog\_socket) \cup P_{Socket}$

- $P(netlink\_xfrm\_socket) = \{nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_xfrm\_socket) = P(netlink\_xfrm\_socket) \cup P_{Socket}$

- $P(netlink\_selinux\_socket) = \{enqueue\}$
- $P(netlink\_selinux\_socket) = P(netlink\_selinux\_socket) \cup P_{Socket}$

- $P(netlink\_audit\_socket) = \{nlmsg\_relay, nlmsg\_readpriv, nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_audit\_socket) = P(netlink\_audit\_socket) \cup P_{Socket}$

- $P(netlink\_ip6fw\_socket) = \{nlmsg\_write, nlmsg\_read\}$
- $P(netlink\_ip6fw\_socket) = P(netlink\_ip6fw\_socket) \cup P_{Socket}$

- $P(netlink\_dnrt\_socket) = \{enqueue\}$
- $P(netlink\_dnrt\_socket) = P(netlink\_dnrt\_socket) \cup P_{Socket}$

$P(dbus) = \{acquire\_svc, send\_msg\}$
$P(nscd) = \{gethost, getstat, getgrp, shmemhost, shmempwd, getpwd, shmemgrp, admin\}$
$P(association) = \{sendto, recvfrom\}$

- $P(netlink\_kobject\_uevent\_socket) = \{enqueue\}$
- $P(netlink\_kobject\_uevent\_socket) = P(netlink\_kobject\_uevent\_socket) \cup P_{Socket}$

El conjunto A tiene como elementos a todos los atributos para los tipos.

Se tiene un conjunto de atributos que se aplican a los tipos que se definen como dominios:
$A_{Domains} = \{domian, daemon, privuser, privrole, userspace_objmgr,$
$priv\_system\_role, privowner, privlog, privmodule, privmem, privfd, privhome,$
$auth, auth\_write, auth\_chkpwd, change\_context, etc\_writer, sysctl\_kernel\_writer,$
$sysctl\_net\_writer, sysctl\_type, admin, userdomain, user\_mini\_domain, mini\_pty\_type,$
$server\_pty, userpty\_type, user\_tty\_type, user\_crond\_domain, user\_home\_dir\_type,$
$gphdomain, fs\_domain, userhelperdomain\}$
De igual manera se tiene un conjunto de atributos para los tipos que se definen como archivo, red, seguridad, PaxFlags y Compat:

$A_{FileTypes} = \{file\_type, device\_type, proc\_fs, dev\_fs,$
$sysadmnfile, fs\_type, exec\_type, tmpfile, user\_tmpfile, xserver\_tmpfile,$
$tmpfsfile, home\_type, home\_dir\_type, ttyfile, ptyfile, pidfile\}$
$A_{NetworkTypes} = \{socket\_type, port\_type, reserved\_port\_type,$
$netif\_type, netmsg\_type, node\_type, logfile, lockfile\}$
$A_{SecurityTypes} = \{login\_contexts, user\_mail\_domain, privmail,$
$user\_home\_type, mta\_user\_agent, mta\_delivery\_agent, mail\_server\_sender,$

$mail\_server\_domain, web\_client\_domain, xserver, xclient, xextension, xproperty,$
$noexattrfile, usercanread, serial\_device, unrestricted, nscd\_client\_domain,$
$nscd\_shmem\_domain, httpdcontent, transitionbool, customizable\}$
$A_{PaXFlags} = \{nopageexec, nosegmexec, nomprotect, norandmmap,$
$noemutramp, norandexec\}$
$A_{Compat} = \{root\_dir\_type, homedirfile\}$

El conjunto A se obtiene de la union de los conjuntos previos de atributos:
$A = A_{Domains} \cup A_{FileTypes} \cup A_{NetworkTypes \cup A_{PaXFlags} \cup A_{Compat}}$

Los conjuntos para roles y usuarios pueden variar de una poítica a otra y son definidos en base a la especificación particular de la política.

Apéndice C

Sistema Kerberos

El sistema Kerberos es utilizado para obtener seguridad en el proceso de actualización de políticas en las fases de autenticación, autorización que involucran protocolos y sistemas cripttograficos. El sistema Kerberos [Steiner88] fue inicialmente implementado por el Instituto Tecnológico de Masashussets en el proyecto Athena [Miller87], ofreciendo una alternativa de autenticación de servicios mediante el modelo presentado por Needham Schoreder [Needham78] y la verificación de identidades.

El sistema Kerberos se conforma de dos servidores de confianza y una base de datos que contiene las contraseñas de los usuarios. Un servidor es llamado Servidor de Autenticación (AS)[1] y el segundo como el Servidor de Boletos (TGS)[2] (ver Figura C.1).

El sistema Kerberos emplea al protocolo nombrado Kerberos, el cual integra las fases de Autenticación, Autorización y Servicio, descritas a continuación.

Protocolo Kerberos V

- Autenticación

1. $A \rightarrow$ AS : A, *TGS*, *T1*

2. AS $\rightarrow A : \{authK,\ TGS,\ Ta\}_{Ka}$, { A, *TGS*, *authK*, *Ta*} $_{Ktgs}$

- Autorización

[1] por las siglas en Inglés *Authentication Server*
[2] por las siglas en Inglés *Ticket Granting Server*

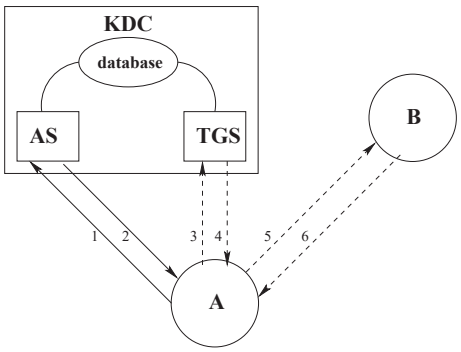

Figura C.1: Estructura de Kerberos

3. $A \rightarrow TGS : \{A,\ TGS,\ authK,\ Ta\}_{Ktgs}, \{A,\ T2\}_{authK}, B$
4. TGS $\rightarrow A : \{servK,\ B,\ Ts\}_{authK}, \{A,\ B,\ servK,\ Ts\}_{Kb}$

- Servicio

5. $A \rightarrow$ B : $\{A,\ B,\ servK,\ Ts\}_{Kb}, \{A,\ T3\}_{servK}$
6. $B \rightarrow$ A : $\{T3 + 1\}_{servK}$

En la Figura C.1, cada número está asociado al mensaje del protocolo Kerberos V. Para la fase de Autenticación, un usuario le proporciona a una entidad A el par *nombre de usuario - contraseña* y la entidad A le envía el mensaje 1 al servidor de Autenticación informándole su identidad, la entidad de interés, en este caso el servidor de Boletos, y una marca de tiempo. El servidor de autenticación le envía a la entidad A el mensaje 2, el cual está formado por dos segmentos que contienen una clave común que sera utilizada por la entidad A y el servidor de Boletos en la fase de autorización.

Para la fase de autorización, la entidad A le envía al servidor de autenticación el mensaje 3, que consta de un segmento del mensaje 2, además de informarle su identidad, y la entidad de interés. El servidor de Boletos le envía el mensaje 4 a la entidad A, en donde se tiene un segmento para la entidad B y otro para la entidad A. Tales segmentos tienen una clave común que será utilizada en la fase de servicio por las entidades A y B.

En la fase de servicio, la entidad A le envía a la entidad B el mensaje 5, formado por un segmento para la entidad B que contiene la clave común que deben utilizar las

entidades A y B. Otro segmento del mensaje 5 contiene la información de la entidad A y una marca de tiempo, tal segmento es cifrado con la clave común entre las entidades A y B.

Actualmente no se conoce ninguna debilidad del protocolo Kerberos V, lo cual constituye la principal razón para emplearlo. Además de poder aprovechar a futuro la característica de integración de sistemas heterogéneos. Para hacer uso del sistema Kerberos además se contar con los servicios kerberizados tales como ssh, scp, telnet, etc. Para nuestro caso solo empleamos al servicio kerberizado de ssh.

Apéndice D

Código del Servidor y Cliente

D.1. Código del Servidor

segmentation.sh

```
#!/bin/bash
#funcion para contar lineas de archivo location
lineslocation(){

lines=‘wc -l $file‘;
# Longitud de la cadena "lines", se le resta 1 para 0 to lines - 1
n=${#lines};

let "n -=1";
#Obtencion posicion de whitespace en "ws"
for i in ‘seq 0 $n‘;
do
if [ "${lines:$i:1}" == " " ]
then
ws=$i;
break;
fi
done
sl=${lines:0:$ws};
}

createlocationwp()
{
if [ -d $dir/$location ]
then
:
else
‘mkdir $dir/$location‘;
echo "Creando politica base localidad $location"
‘cp -r policy/* $dir/$location/‘;
u="$dir/$location/users";
‘touch $u‘;
fi

nf=‘echo $rule | awk ’/^user/{print NF}’‘;
case $nf in
```

```
6)
‘echo $rule | awk ’/^user/{print $1, $2, $5,$6}’ >> $u‘;
;;
7)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7}’ >> $u‘;
;;
8)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8}’ >> $u‘;
;;
9)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9}’ >> $u‘;
;;
10)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10}’ >> $u‘;
;;
11)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10, $11}’ >> $u‘;
;;
12)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10, $11, $12}’ >> $u‘;
;;
13)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10, $11, $12, $13}’ >> $u‘;
;;
14)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10, $11, $12, $13, $14}’ >> $u‘;
;;
15)
‘echo $rule | awk ’/^user/{print $1, $2, $5, $6, $7, $8, $9, $10, $11, $12, $13, $14, $15}’ >> $u‘;
;;
esac
}

descentralized()
{
lineslocation;
for i in ‘seq 1 $sl‘;
do
if [[ $i -ge 1 && $i -le $sl ]]
then

if [ $i -eq 1 ]
then
rule=‘head -1 $file‘;
fi

if [ $i -eq $sl ]
then
rule=‘tail -1 $file‘;
fi

if [[ $i -gt 1 && $i -lt $sl ]]
then
rule=‘head -$i $file | tail -1‘;
fi

r1=‘echo $rule | awk ’/^user/{print $1}’‘;
r2=‘echo $rule | awk ’/^user/{print $3}’‘;
nr1=${#r1};
nr2=${#r2};
if [[ $r1 = "user" && $nr1 -gt 0 && $r2 = "location" && $nr2 -gt 0 ]]
then
location=‘echo $rule | awk ’/^user/{print $4}’‘;
```

```
createlocationwp;
fi
fi

done
}

compile()
{
l="$dir/$location";
echo "Compilando politica localidad $location"
‘cat $l/flask/security_classes $l/flask/initial_sids $l/flask/access_vectors > $l/p.tmp‘;
‘cat $l/tunables/*.tun >> $l/p.tmp‘
‘cat $l/attrib.te $l/tmp/program_used_flags.te >> $l/p.tmp‘;
‘cat $l/macros/program/*.te  $l/macros/*.te >> $l/p.tmp‘;
‘cat $l/types/*.te >> $l/p.tmp‘;
‘cat $l/domains/*.te  $l/domains/misc/*.te $l/domains/program/*.te $l/assert.te $l/rbac $l/users >> $l/p.tmp‘;
‘cat $l/constraints >> $l/p.tmp‘;
‘cat $l/initial_sid_contexts $l/fs_use $l/genfs_contexts $l/net_contexts >> $l/p.tmp‘;
‘m4 -Imacros -s $l/p.tmp > $l/policy.tmp‘;
‘mv $l/policy.tmp $l/policy.conf‘;
resultado=‘checkpolicy $l/policy.conf -o $l/policy.18‘;
echo "policy ok"
echo ""
h=‘md5sum $l/policy.18‘;
b=${h:0:32};
‘echo "$location $b" >> $md5‘;
}

genpolicy()
{
lineslocation;
for i in ‘seq 1 $sl‘;
do
if [[ $i -ge 1 && $i -le $sl ]]
then

if [ $i -eq 1 ]
then
rule=‘head -1 $file‘;
fi

if [ $i -eq $sl ]
then
rule=‘tail -1 $file‘;
fi

if [[ $i -gt 1 && $i -lt $sl ]]
then
rule=‘head -$i $file | tail -1‘;
fi
r1=‘echo $rule | awk ’/^location/{print $1}’‘;
nr1=${#r1};
if [[ $r1 = "location" && $nr1 -gt 0 ]]
then
location=‘echo $rule | awk ’/^location/{print $2}’‘;
compile;
fi
fi
done
}

echo ""
```

```
echo "Destruyendo Boletos"
destroy=‘kdestroy‘;
echo ""
echo "Creando Boleto"
destroy=‘kinit < .algo‘;
echo "ok"

#variables para location & locations
dir="./locations";
file="./location";
md5="newmd5"

#eliminar y crear directorio locations
‘rm -rf $dir‘;
‘rm -rf $md5‘;
‘mkdir $dir‘;
‘touch $md5‘;

echo ""
descentralized;
echo ""
genpolicy;
./cinco.sh
```

siete.sh

```
#!/bin/bash
uno=‘checkpolicy -o salida‘;
dos=‘load_policy -b salida‘;
```

D.2. Código del Cliente

client.sh

```
#!/bin/bash
localidad=‘hostname -s‘;
echo "Inicio del proceso de Verificacion y Actualizacion de politica localidad $localidad";
echo ""
echo "Destruyendo Boletos"
destroy=‘kdestroy‘;
echo "Creando Boleto"
destroy=‘kinit < .algo‘;
echo ""
echo "ok"
echo ""
md5i386=‘cat siete.sh | ssh -K i386 /bin/bash‘;
md5sum=‘md5sum policy.conf‘;
md5=${md5sum:0:32};
echo "Valor md5 politica localidad $localidad";
echo "lado servidor:";
echo $md5i386;
echo "lado cliente:"
echo $md5;
if [ $md5i386 = $md5 ]
then
  echo "politicas iguales entre servidor y localidad $localidad"
else
  echo "politicas diferentes entre servidor y localidad $localidad"
  echo "actualizando la politica de la localidad $localidad"
 ‘cat ocho.sh | ssh -K i386 /bin/bash‘;
```

```
  echo "compilando politica de la localidad $localidad"
uno=‘checkpolicy -o salida‘;
  echo "cargando politica en el kernel de la localidad $localidad"
dos=‘load_policy -b salida‘;
fi
echo ""
echo "Fin del proceso de Verificacion y Actualizacion de politica localidad $localidad";
```

siete.sh

```
#!/bin/bash
h=‘md5sum /root/selinux/bash/tmp/locations/amd64/policy.conf‘;
b=${h:0:32};
echo $b
```

ocho.sh

```
#!/bin/bash
‘scp /root/selinux/bash/tmp/locations/amd64/policy.conf amd64:‘;
```

Apéndice E

Instalación de SELinux

Instalación de SELinux en la distribución de Linux Gentoo.

Obtención de los fuentes endurecidos que incluyen las librerias de SELinux:

```
emerge -av hardened-sources
```

Se debe configurar y reconstruir al kernel para integrar los parametros de operación de SELinux (etiquetas de seguridad del sistema de archivos, soporte para desarrollo, etc).

```
General setup  --->
     [*] Prompt for development and/or incomplete code/drivers
     [*] Auditing support
     [*]   Enable system-call auditing support

File systems --->
     <*> Second extended fs support (If using ext2)
     [*]   Ext2 extended attributes
     [ ]     Ext2 POSIX Access Control Lists
     [*]     Ext2 Security Labels
     <*> Ext3 journalling file system support (If using ext3)
     [*]   Ext3 extended attributes
     [ ]     Ext3 POSIX Access Control Lists
     [*]     Ext3 Security labels
     <*> JFS filesystem support (If using JFS)
     [ ]   JFS POSIX Access Control Lists
     [*]   JFS Security Labels
     [ ]   JFS debugging
     [ ]   JFS statistics
     <*> XFS filesystem support (If using XFS)
     [ ]   Realtime support (EXPERIMENTAL)
     [ ]   Quota support
     [ ]   ACL support
     [*]   Security Labels
     Pseudo Filesystems --->
          [*] /proc file system support
          [*] Virtual memory file system support (former shm fs)
```

```
Security options  --->
     PaX  --->
          [*] Enable various PaX features
                PaX Control  --->
                     [ ] Support soft mode
                     [*] Use legacy ELF header marking
                     [*] Use ELF program header marking
                         MAC system integration (none)  --->
                Non-executable pages  --->
                     [*] Enforce non-executable pages
                     [*]   Paging based non-executable pages
                     [*]   Segmentation based non-executable pages
                     [*] Emulate trampolines
                     [*] Restrict mprotect()
                     [ ]   Disallow ELF text relocations
                Address Space Layout Randomization  --->
                     [*] Address Space Layout Randomization
                     [*]   Randomize kernel stack base
                     [*]   Randomize user stack base
                     [*]   Randomize mmap() base
                     [*]     Randomize ET_EXEC base
   [*] Enable different security models
   [*]   Socket and Networking Security Hooks
   <*>   Default Linux Capabilities
   [*] NSA SELinux Support
   [ ]   NSA SELinux boot parameter
   [ ]   NSA SELinux runtime disable
   [*]   NSA SELinux Development Support
   [ ]   NSA SELinux AVC Statistics
   (1)   NSA SELinux checkreqprot default value
   [ ]   NSA SELinux enable new secmark network controls by default
   [ ]   NSA SELinux maximum supported policy format version
```

Adicionar en el erchivo /etc/fstab la línea para el sistema de archivos de SELinux:

```
...
none                    /selinux        selinuxfs       defaults        0 0
```

Instalación de las políticas:

```
emerge -av checkpolicy policycoreutils FEATURES=-selinux emerge -av selinux-base-policy
```

La política esta contenida en el directorio:

```
cd /etc/selinux/src/policy/
```

El archivo de la política (policy.conf), es compilado con make, la herramienta de generación o automatización de código. La opción load genera al archivo de la política, compila y la carga al kernel. Por otra parte la opción policy genera al archivo de la política y la compila solamente.

Referencias

[Abendroth03a] Abendroth, J. y Jensen, C. D. A unified security mechanism for networked applications. March 2003. In Proccedings of 18th Symposium on Applied Computing (SAC2003).

[Abendroth03b] Abendroth, O. y Jensen, C. D. Partial outsourcing: A new paradigm for access control. June 2003. SACMAT 03 Como Italy.

[Al-Kahtani03] Al-Kahtani, M. A. y Sandhu, R. Induced role hierarchies with atribute-based rbac. June 2003. SACMAT 03 Como Italy.

[Anderson72] Anderson, J. P. Computer security technology planning study. Inf. téc., 1972. Volume I,II.

[Badger95] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., y Haghighat, S. A. A domain and type enforcement unix prototype. June 1995. Unix Security Symposium Proceedings.

[Baker97] Baker, D. Pcasso: Appliying and extending state-of-the-art security in the healthcare domain. 1997. Proceedings of the Annual Computer Security Applications Conference.

[Bell73] Bell, D. E. y LaPadula, L. Secure computer systems: Mathematical foundations. Inf. téc., 1973. MITRE Technical Report 2547, Volume I.

[Bertino05] Bertino, E., Catania, B., y Damiani, M. L. Geo-rbac: A spatially aware rbac. June 2005. SACMAT 05 Stockholm Sweden.

[Boebert85] Boebert, W. E. y Kain, R. A practical alternative to hierarchical integrity policies. *Proc. of the 8th National Computer Security Conference, Gaithersburg, MD*, 1985.

[Bwever89] Bwever, D. F. y Nash, M. J. China wall security policy. May 1989. IEEE Symposium on Research in Security and Privacy.

[Bykova04] Bykova, M. y Atallah, M. Succinct specifications of portable document access policies. June 2004. SACMAT 04 New York USA.

[Chadwick02] Chadwick, D. W. y Otenko, A. The permis x.509 role based privilege management infraestructure. June 2002. SACMAT 02 Monterrey California USA.

[Ch.L.Pedro06] Ch.L.Pedro, J.J.Flores, y J.M.G.Garcia. User profile linear correlation and its use on intrusion detection. 2006. Mexican Conference on Informatics Security, Mexico, D.F.

[Ch.L.Pedro07] Ch.L.Pedro, J.J.Flores, y J.M.G.Garcia. User profile linear correlation and its use on intrusion detection. 2007. Workshop of Computer Security on the Mexican International Conference on Artificial Inteligence, Aguascalientes, Aguascalientes, Mex.

[Ch.L.Pedro08a] Ch.L.Pedro, J.J.Flores, y J.M.G.Garcia. An architecture for systematic administration of selinux policies in distributed environments. 2008. International Journal of Computers and Communications, United Kingdom.

[Ch.L.Pedro08b] Ch.L.Pedro, J.J.Flores, y J.M.G.Garcia. Arquitectura de seguridad para la sistematizada politicas selinux en ambientes distribuidos. 2008. Congreso Nacional de Ingenieria y Arquitectura IA08, Morelia, Mich., Mex.

[Ch.L.Pedro08c] Ch.L.Pedro, J.J.Flores, y J.M.G.Garcia. Security architecture for a systematic administration of selinux policies in distributed environ-

ments. 2008. Recent Advances in Data Networks, Communications, Computers, Bucharest, Romania.

[Clark87] Clark, D. D. y Wilson, D. R. A comparison of commercial and military computer security policies. April 1987. IEEE Symposium on Computer Security and Privacy.

[Cohen02] Cohen, E., Winsborough, W., y Shands, R. K. T. D. Models for coallition based access control (cbac). June 2002. SACMAT 02 Monterrey California USA.

[Coker05] Coker, F. y Coker, R. Taking advantadge of selinux. 2005.

[Cokerrg] Coker, F. Getting started with selinux howto: the new selinux, 2004, note=http://lurking-grue.org.

[cpd07] Código penal para el distrito federal. Inf. téc., Febreaury 2, 2007. Http://www.paot.org.mx/centro/codigos/df/pdf/cpdfn.pdf.

[cpf09] Código penal federal. Inf. téc., January 23, 2009. Http://www.cddhcu.gob.mx/LeyesBiblio/pdf/9.pdf.

[cps06] Código penal para el estado de sinaloa. Inf. téc., August 11, 2006. Http://www.stj-sin.gob.mx/Leyes/CODPENAL.html.

[dod85a] Department of defense trusted computer system evaluation criteria. Inf. téc., Departament of Defense, December 1985. DoD 5200.28-STD.

[DOD85b] DOD. Departament of defense trsuted computer system evaluation criteria. Inf. téc., Departament of Defense, 1985.

[Drimer07] Drimer, S. y Murdoch, S. J. Keep your enemies close: Distance bounding against smartcard relay attacks. 2007. 16th USENIX Security Symposium.

[Ferraiolo92] Ferraiolo, D. y Khun, R. Role-based access control. 1992. Proceedings 15th National Computer Security Conference.

[Frank Mayer06] Frank Mayer, K. M. y Caplan, D. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006.

[F.Wedde03] F.Wedde, H. y Lischka, M. Cooperative role-based administration. June 2003. SACMAT 03 Como Italy.

[Gasser88] Gasser, M. *Building a Secure Computer System*. Van Nostrand Reinhold, 1988.

[Group92] Group, N. W. Rfc 1321 the md5 message-digest algorithm. Inf. téc., April 1992. Http://www.faqs.org/rfcs/rfc1321.html.

[Grunbacher03] Grunbacher, A. Posix access control lists on linux. Inf. téc., 2003. Suse Administration Guide.

[Hallyn97] Hallyn, S. E. Domain ant type enforcement for linux. 1997.

[Harrington03] Harrington, A. y Jensen, C. Cryptographic access control in a distributed file system. June 2003. SACMAT 03 Como Italy.

[hei09] Heimdal implementation of kerberos 5. 2009. Http://www.h5l.org/.

[Hengartner04] Hengartner, U. y Steenkiste, P. Implementing access control to people location information. June 2004. SACMAT 04 New York USA.

[Herzog05] Herzog, A. L., Guttman, J. D., Harris, D. R., Ramsdell, J. D., Segall, A. E., y Sniffen, B. T. Policy analysis and generation work at mitre. 2005. Security Enhanced Linux Symposium.

[iso08] Iso/iec 15408. information technology - security techniques - evaluation criteria for it security. Inf. téc., International Organization for Standardization, 2008. Parts 1,2,3.

[J.77] J., B. K. Integrity considerations for secure computer systems. Inf. téc., 1977. MITRE Technical Report 3153.

[Jaeger02] Jaeger, T., Edwards, A., y Zhang, X. Managing access control policies using access control. June 2002. SACMAT 02 Monterrey California USA.

[Jaeger04] Jaeger, T., reiner Sailer, y Zhang, X. Resolving constraint conflicts. June 2004. SACMAT 04 New York USA.

[Joshi03] Joshi, J. B. D., Bertino, E., Shafiq, B., y Ghafoor, A. Dependencies and separation of duty constraints in gtrbac. June 2003. SACMAT 03 Como Italy.

[Jupiter03] Jupiter, S. For more information about sam jupiter. 2003. Http://www.sam-security.com.

[Kern02] Kern, A., Kuhlmann, M., Schaad, A., y Moffett, J. Observations on role life-cycle in the context of enterprise security management. June 2002. SACMAT 02 Monterrey California USA.

[Kern03] Kern, A., Schaad, A., y Moffett, J. An administration concept for the enterprise role-based access control model. June 2003. SACMAT 03 Como Italy.

[Kuhlmann03] Kuhlmann, M. y Schimpf, G. Role mining-revealing business roles for security administration using data mining technology. June 2003. SACMAT 03 Como Italy.

[Lampson71] Lampson, B. W. Protection. *Proc. 5th Princeton Symp. Information Sciences and Systems*, 1971.

[Landwehr81] Landwehr, C. E. Formals models for computer security. *Computer Surverys Vol. 13 ACM*, 1981.

[Longstaff03] Longstaff, J., Lockyer, M., y Nicholas, J. The tees confidentiality model: an authorisation model for identities and roles. June 2003. SACMAT 03 Como Italy.

[Loscoco00] Loscoco, P., Smalley, S., Muckelbauer, P. A., y Taylor, R. C. The inetability of failure: The flawed assumption of the security in moderm computer enviroments. 2000. NSA.

[Loscoco01] Loscoco, P. y Smalley, S. Integrating flexible support for security policies into linux operating system. 2001. NSA.

[lpd03] Ley de protección de datos personales del estado de colima. Inf. téc., June 14, 2003. Http://www.ucol.mx/radio/textos/sip-4753.pdf.

[Lucas Ballard06] Lucas Ballard, D. L., Fabian Monrose. Biometric authentication revisited: Understanding the impact of wolves in sheep's clothing. 2006. 15th USENIX Security Symposium.

[McCarty04] McCarty, B. *SELinux NSA's Open Source Security Enhanced Linux*. O'Reilly, 2004.

[Miller87] Miller, S. P., Schiller, N. J. I., y Saltzer, J. H. Kerberos authentication and authorization system. December 21, 1987. M.I.T. Project Athena, Cambridge, Massachusetts.

[mit09] Mit implementation of kerberos 5. 2009. Http://web.mit.edu/kerberos.

[Nakamura05] Nakamura, Y. Simplifying policy management with selinux policy editor. 2005. Security Enhanced Linux Symposium.

[Needham78] Needham, R. M. y Schoroeder, M. D. Using encryption for authentication in large networks of computers. December 1978. Communications of the ACM, Vol. 21 (12), pp. 993-999.

[Nyanchama94] Nyanchama, M. y Osborn, S. L. Access rights administration in role-based security systems. 1994. Database Security, VIII, Status and Prospects WG11.3 Working Conference on Database Security.

[ofStandards93] of Standards, N. I. y Technology. Fips pub 180-1 secure hash standard. Inf. téc., May 11, 1993. Http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[Osborn02] Osborn, S. L. Information flow analysis of an rbac system. June 2002. SACMAT 02 Monterrey California USA.

[Osborn03] Osborn, S. L., Han, Y., y Liu, J. A methodology for managing roles in legacy systems. June 2003. SACMAT 03 Como Italy.

[Park02] Park, J. y Sandhu, R. Towards usage control models: Beyond traditional control. June 2002. SACMAT 02 Monterrey California USA.

[pdm08] Selinux policy management and distribution. 2008. Http://oss.tresys.com/projects/policy-server/wiki/PmdPrototype.

[Phillips02] Phillips, C. E., Ting, y Demurjian, S. A. Information sharing and security in dynamic coalitions. June 2002. SACMAT 02 Monterrey California USA.

[R.99] R., S., S.D., S., P., L., M., H., D., A., y J., L. The flask security architecture: System support for diverse security policies. August 1999. Proceedings of the 8th USENIX Security Simposium.

[Ray02] Ray, I., Ray, I., y Narasimhamurthi, N. A cryptographic solution to implement control in a hearchy and more. June 2002. SACMAT 02 Monterrey California USA.

[Red05] Red Hat, Inc., PO Box 13588 Research Triangle Park NC 27709 USA. *Red Hat Enterprise Linux 4 SELinux Guide*, 2005. Http://www.redhat.com.

[Sandhu01] Sandhu, R., Ferriaiolo, D., Gavrila, S., Kuhn, D. R., y Chandramoulli, R. Proposed nist standart for role-access control. 2001. ACM Transactions on Information and System Security Vol. 4 No. 3.

[sel09] Selinux, 2009. Http://www.nsa.gov/selinux.

[SELinux06] SELinux. Selinux symposium, 2006. Http://www.selinux-symposium.org.

[Silberschatz05] Silberschatz, Galvin, y Gagne. *Operating Systems Concepts*. John Wiley and Sons, seventh ed$\underline{\text{ón}}$., 2005.

[Smith05] Smith, P. G. *Linux Network Security*. Charles River Media, 2005.

[Sonia Chiasson06] Sonia Chiasson, P. v. O. y Biddle, R. A usability study and critique of two password managers. 2006. 15th USENIX Security Symposium.

[Steiner88] Steiner, J. G., Neuman, C., y Schiller, J. I. Kerberos: An authentication service for open networks systems. February, 1988. Usenix Conference Proceedings, Dallas, Texas, pp. 191-202.

[Stephen200505] Stephen2005. Configuring the selinux policy. Inf. téc., 2005.

[Swanson96] Swanson, M. y Guttman, B. Generally accepted principles and practices for securing information technology systems. Inf. téc., National Institute of Standards and Technology, September 1996.

[ter05] Type enforcement rules and macros, 2005. A primer course for Linux Engineers.

[Toxen00] Toxen, B. *Real World Linux Security*. PTR, 2000.

[tre05] A primer course for linux engineers, 2005. Http://www.tresys.com/.

[typ05] Types and attributes, 2005. A primer course for Linux Engineers.

[typ09] Type enforcement technology, 2009. Http://www.securecomputing.com.

[Valentin Sgarciu06] Valentin Sgarciu, M. S. V. Smart card technology used in secured personal identification systems. Bucharest, Romania, 2006. Proceedings

of the 5th WSEAS Int. Conf. on DATA NETWORKS, COMMUNICATIONS & COMPUTERS.

[Walker96] Walker, K. M., Sterne, D. F., Badger, M. L., Petkac, M. J., Sherman, D. L., y Oostendorp, K. A. Confining root programs with type enforcement (dte). 1996. Sixth USENIX UNIX Security Symposium.

[Walsh03] Walsh, D. J. Selinux targeted vs strict policy. 2003.

[Walsh05] Walsh, D. Targeted vs strict policy history and strategy. 2005. Security Enhanced Linux Symposium.

[Zanin04] Zanin, G. y Mancini, L. V. Towards a formal model for security policies specification and validation in the selinux system. June 2004. SACMAT 04 New York USA.

[Zhang02] Zhang, L., Ahn, G. J., y Chu, B. T. A role based delegation framework for healthcare. June 2002. SACMAT 02 Monterrey California USA.

[Zhang04] Zhang, X., Park, J., y Presicce, F. P. A logical specification for usage control. June 2004. SACMAT 04 New York USA.