Treball fi de carrera

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques
Universitat de Barcelona

MODELS DE SEGURETAT AL NUCLI Linux 2.6

Pau Varela Navarro

Director: Jaume Timoneda Salat
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 21 de setembre de 2005

ÍNDEX

1.- Introducció

Paral·lela a l'ús cada cop més massiu dels ordinadors, avança la necessitat d'incrementar la seguretat en els nostres sistemes informàtics. Cada dia són més les màquines connectades a Internet les 24 hores del dia, set dies a la setmana i 365 dies l'any, completament accessibles des de qualsevol ordinador de la Xarxa. Protegir tant els serveis oferts a l'exterior com els recursos i les dades, compartides o no, dels usuaris d'un sistema informàtic esdevé, per tant, crucial.

La seguretat als sistemes informàtics engloba tant la vessant del maquinari com la del programari. En el primer cas es tractarà d'assegurar l'accés físic a la màquina, i oferir unes condicions mínimes que en garanteixin la seva integritat física, com poden ser els mecanismes per evitar pèrdues produïdes per talls elèctrics o baixades de tensió sobtades, ventilació i manteniment de temperatures adequades pel maquinari, el control d'incendis i la seguretat en l'accés a les instal·lacions on tenim els nostres sistemes informàtics, o l'accés a components definides pels fabricants del maquinari que estiguem utilitzant, com ara la BIOS (*Basic Input/Output System*). Les qüestions relacionades amb la seguretat física no seran tractades en aquest treball.

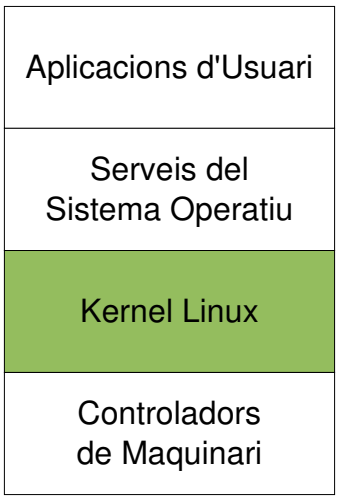
Pel que fa a la seguretat per la banda del programari, l'àmbit d'acció és més ampli, i comprèn des de la prevenció fins a la recuperació de possibles desastres. Alguns dels aspectes més destacats d'aquest tipus de seguretat són:

- Gestió d'usuaris i grups
- Gestió de contrasenyes i sistemes de xifratge
- Control d'accés a fitxers i gestió del sistema de fitxers
- Seguretat a la xarxa
- Seguretat al nucli del sistema operatiu
- Generació i seguiment de *logs* del sistema
- Detecció d'intrusos i antivirus
- Polítiques de còpies de seguretat

En aquest llistat hi ha unes característiques que estan directament lligades amb el funcionament intern del sistema operatiu i el nucli que el gestiona, i n'hi ha d'altres que solen ser implementades en forma d'aplicacions d'usuari que, tot i ser desenvolupades per a determinats sistemes operatius, no tenen a veure amb el seu funcionament en general. Existeixen un munt d'aplicacions per reforçar, analitzar i realitzar informes que ens ajudin a comprendre l'estat de la seguretat del nostres sistemes informàtics, així com per crear rèpliques de les dades que contenen. L'estudi d'aquesta mena d'aplicacions d'usuari tampoc entra dins dels objectius d'aquest treball.

Nosaltres ens centrarem en els aspectes de la seguretat informàtica relacionats amb el sistema operatiu GNU/Linux i, més concretament, en aquells que fan referència al seu *kernel* o nucli i les operacions que depenen d'ell.

El sistema operatiu GNU/Linux, com el seu nom indica, consta de dues parts principals: el sistema GNU[1] (*GNU's Not UNIX*), que ens proporciona un seguit d'utilitats d'usuari i de sistema, i un nucli Linux, que permet la interacció entre aquestes utilitats i el maquinari que utilitza un sistema informàtic. El següent esquema representa les diferents capes que conformen el sistema operatiu objecte d'estudi:



Les **aplicacions d'usuari** són les que en última instància usem les persones que fem servir els equips informàtics i van des de programes per l'edició de text, navegadors o clients de correu electrònic, fins a reproductors d'àudio i vídeo, programari per realitzar còpies de seguretat i un innombrable etcètera que creix amb les necessitats dels usuaris.

Els **serveis del sistema operatiu** són aquells que són considerats típicament com a part del mateix sistema operatiu, com poden ser l'entorn gràfic de finestres, la consola de comandes o *shell*, així com els compiladors i les llibreries que ens permeten accedir a la interfície del kernel.

1 http://www.gnu.org

El **kernel Linux** ja hem vist que és qui abstrau i gestiona l'accés als recursos de maquinari, incloent l'accés al processador, memòria i dispositius d'emmagatzematge.

Els **controladors de maquinari** són aquells que ens permeten comunicar-nos directament amb cadascun dels dispositius físics connectats al nostre sistema com poden ser els dispositius de xarxa, so, imatge, o la pròpia memòria RAM i el processador, entre molts d'altres.

Segons aquest esquema, cadascuna de les capes és capaç de comunicar-se amb les que li són immediatament adjacents, i la dependència entre les capes va de dalt a baix, de manera que les capes superiors necessiten de les inferiors per poder funcionar, mentre que les inferiors poden funcionar de forma independent.

D'aquestes capes, la que ens interessa és la del kernel, que podem considerar-la com la suma de cinc components principals:

El **planificador de processos** és l'encarregat de gestionar l'accés dels processos a la CPU. Aquest planificador o *scheduler* pot seguir diferents algoritmes per implementar una política o una altra, però és qui decideix quins processos poden entrar a consumir recursos de la CPU, i garanteix que tots disposin d'un cert temps de processador i no morin d'inanició.

El **gestor de memòria** o *memory manager* (MM) és qui reparteix l'espai de memòria principal del sistema entre els diferents processos que poden córrer simultàniament. A més, el gestor suporta memòria virtual, que permet oferir als processos més memòria de la que en realitat té físicament el sistema, i gestiona la memòria d'intercanvi o *swap*, que guarda les dades no utilitzades de la memòria en dispositius d'emmagatzematge secundari segons les necessitats del moment.

El **sistema de fitxers virtual** (VFS) és l'encarregat de fer una abstracció de tots els tipus de dispositius físics, oferint un interfície comuna per tots els dispositius. L'VFS és també qui dóna suport als diferents sistemes de fitxers existents.

La **interfície de xarxa** proporciona accés a uns quants protocols de xarxa, així com a dispositius físics que ens hi permeten treballar.

Amb la **comunicació entre processos** (IPC) se'ns ofereixen una sèrie de mecanismes que permeten als processos intercanviar dades o informació entre ells. Per defecte, el kernel Linux utilitza el System V IPC[1].

1 http://en.wikipedia.org/wiki/System_V

• Objectius:

Un cop definit quin serà el nostre objecte d'estudi fixarem ara quins són els objectius d'aquest treball: en primer lloc, fer una anàlisi de les noves funcionalitats en matèria de seguretat que aporta la versió 2.6 del kernel Linux, així com de la seva implementació. En segon lloc, volem mostrar quins models de seguretat són distribuïts amb les fonts del nucli i quin és el seu funcionament i característiques principals.

Per poder concretar una mica més aquests objectius, necessitem descriure el marc en el qual neixen les necessitats d'aportar millores en la seguretat del nucli. Per què si bé la seguretat al kernel sempre s'ha considerat important, és amb l'ús intensiu dels sistemes informàtics quan apareixen els *bugs,* o errades de disseny, que permeten l'explotació de vulnerabilitats o que poden provocar estats d'inconsistència al sistema.

A les primeres versions del nucli Linux, la seguretat al kernel es realitzava mitjançant la separació de dos espais d'execució, l'espai d'usuari i l'espai del nucli, i mitjançant el sistema de permisos tradicional dels sistemes UNIX, que proporciona un usuari privilegiat (`root`) a qui no se li apliquen restriccions i la possibilitat de definir múltiples usuaris que es veuen sotmesos al sistema de permisos establert.

Amb el temps, aquest model de "tot o res" va anar perdent vigència ja que davant de comptes de `root` compromesos en possessió d'usuaris que no són administradors, les conseqüències de no tenir cap mena de restriccions podien ser desastroses. Mica en mica van començar a aparèixer esquemes de seguretat com ara el definit al document Linux-privs[1] i, a partir de la versió 2.0 del kernel es va implementar un model similar als nivells de seguretat BSD, els `securelevels`, que permetien l'activació de certs nivells de seguretat que retallaven més o menys el conjunt d'operacions permeses al superusuari `root`.

A la versió 2.2 del kernel, els `securelevels` van ser substituïts per un sistema de *capabilities* o capacitats, que volia implementar l'especificació de la interfície de programació POSIX (*Portable Operating System Interface, uniX based*) definida al document POSIX 1003.1e. Aquest document va servir de base per dividir els privilegis de `root` en un conjunt d'habilitats o capacitats que podem activar o no i que li deixen o impedeixen realitzar determinades operacions. El sistema de capacitats encara és vigent avui en dia i el veurem a la secció 5 d'aquest treball.

A més, des de sempre ha existit la possibilitat d'evitar o corregir vulnerabilitats mitjançant *patchs* o pedaços, que són fragments de codi de s'afegeixen al del propi nucli abans de compilar-lo i que permeten l'aplicació de modificacions al kernel per, per exemple, corregir aquestes vulnerabilitats. La versió 2.4 del kernel incorporava el model de capacitats i cada cop apareixien més *patchs* que implementaven polítiques de seguretat.

1 http://www.linuxsecurity.com/resource_files/server_security/linux-privs/linux-privs.html

L'esperada aparició de la versió 2.6 del nucli, a finals de l'any 2003, va aportar un marc o *framework* de seguretat que permet el desenvolupament de models de seguretat en forma de mòduls del nucli, proporcionant una interfície unificada per desenvolupar polítiques de seguretat. Aquesta interfície va ser desenvolupada pel projecte Linux Security Modules (LSM) i va ser integrada pràcticament en la seva totalitat al nucli (alguns dels aspectes relacionats amb la xarxa no van ser inclosos a la sèrie 2.6).

Un cop vista aquesta breu evolució històrica de la seguretat al kernel, podem concretar una mica més els objectius de nostre treball, que són:

- Fer l'anàlisi d'aquest nou *framework* anomenat LSM.
- Veure quina és la seva implementació al nucli Linux 2.6.
- Realitzar un estudi dels models de seguretat, implementats en forma de mòduls LSM, que s'han incorporat a les fonts del kernel.

L'excusa de l'estudi dels models de seguretat també ens permetrà conèixer més a fons quines són les peculiaritats d'un sistema operatiu que desperta passions.

- **Contingut:**

Començarem el nostre estudi a la secció 2, definint una sèrie de **Conceptes Previs** utilitzats amb freqüència i que estan relacionats amb la seguretat. Així, veurem la definició dels espais d'usuari i de nucli i dels mòduls del kernel, el sistema tradicional de permisos UNIX, què són les matrius d'accés i les diferències entre un model de control d'accés DAC (*Discretionary Access Control)* i un de MAC (*Mandatory Access Control*).

A la secció 3, farem un repàs de les **Noves Funcionalitats** aportades per la branca 2.6 del kernel de Linux, que no són pas poques. Aquesta nova versió del nucli l'ha popularitzat una mica més i la seva integració és una realitat per cada dia més arquitectures i tipus de maquinari, fet que permet el creixement de la seva comunitat d'usuaris.

Abordarem la qüestió dels **Linux Security Modules** a la secció 4, on veurem què són exactament i com s'implementen aquest tipus de mòduls dins del kernel 2.6. En aquesta secció també és on expliquem el funcionament de tres dels cinc models de seguretat que s'incorporen al nou nucli. Separem els dos mòduls més grans en dues seccions independents per poder estructurar millor l'explicació i perquè tenen prou entitat com per què els hi dediquem una secció a cadascun. Aquests mòduls són els que implementen les **Linux Capabilities**, que veurem a la secció 5, i la modificació del kernel **NSA SELinux**, descrita a la secció 6.

No hem volgut oblidar la feina feta per un munt de desenvolupadors que han aportat *patchs* de seguretat al kernel igual de vàlids que els LSM. A la secció 7, potser la més curta, presentem **Altres Maneres** d'aplicar models de seguretat al kernel Linux, fent una breu descripció d'un parell de projectes seguretat prou potents que ens ofereixen una alternativa als LSM. L'estudi no ha estat tant exhaustiu com en el cas dels LSM, inclosos a la branca oficial del kernel, però és una primera aproximació a d'altres maneres existents d'oferir seguretat.

Amb les **Conclusions**, a la secció 8 i la **Bibliografia** a la 9, finalitzarem el nostre treball.

- **Qüestions pràctiques:**

Per tal d'experimentar amb els diferents models de seguretat i comprovar les funcionalitats que aporten, hem utilitzat una màquina de proves. Aquesta màquina, amb un processador AMD Mobile a 1857 MHz i 256 MB de memòria RAM, té instal·lada la distribució Debian GNU/Linux[1] en la seva versió `testing`, i un kernel Linux en la versió 2.6.12.1. És per això que quan parlem de les "versions actuals del nucli" farem referència a aquesta versió del nucli i durant el treball es fa referència a algunes utilitats pròpies de la distribució emprada. Tot i això, quan mostrem exemples, hem procurat utilitzar les instruccions més estàndards possibles.

De la mateixa manera, quan citem fitxers de l'arbre de directoris, les rutes poden canviar en funció de la distribució utilitzada. Per evitar malentesos, hem fet servir la següent nomenclatura: quan parlem d'un fitxer situat a `<security/fitxer.c>` fem referència al contingut dins del directori `/usr/src/linux/`, que és on la distribució Debian GNU/Linux recomana guardar les fonts del nucli. Per tant, la ruta absoluta d'aquest fitxer seria la següent: `/usr/src/linux/security/fitxer.c`. En cas d'aplicar alguna modificació d'aquest criteri, l'anunciarem degudament.

1 http://www.debian.org

2.- CONCEPTES PREVIS:

En aquest apartat farem definirem una sèrie de conceptes que seran utilitzats durant tot el document. Alguns d'aquests conceptes ens resultaran familiars i d'altres no tant, però permeten sentar les bases per a la redacció del document. Els conceptes descrits tenen a veure tant amb la seguretat com amb aspectes fonamentals dels kernels Linux.

● Espai d'Usuari i Espai de Nucli:

Els sistemes UNIX i els seu derivats, com és el cas de GNU/Linux, permeten l'execució de codi a dos espais: l'espai de nucli i l'espai d'usuari. Aquests dos espais o modes tenen associats uns permisos diferents segons el cas, i vénen a implementar un primer nivell de seguretat en el sistema operatiu.

Així, el codi executat a l'espai de nucli s'escapa de qualsevol restricció que pugui imposar el sistema, i pot moure's lliurement sense haver de recórrer a les capes d'abstracció d'accés als diferents medis que ofereix el propi nucli. Treballant en mode nucli tindrem accés a tota la memòria del sistema, així com a les taules de processos (fins i tot per a modificar-les) i a les dades que aquests utilitzen. Qui sol treballar en aquest espai és el propi nucli del sistema operatiu i alguns altres processos de baix nivell de gestió (memòria, sistema de fitxers, ...).

L'espai de nucli també es coneix com a espai de kernel, *kernel-land* o *ring 0*.

Pel que fa a l'espai d'usuari (o *user-land*), és on funcionen la majoria de les aplicacions que puguem exigir-li a un ordinador i disposa d'una sèrie de restriccions, d'entre les que destaquen:

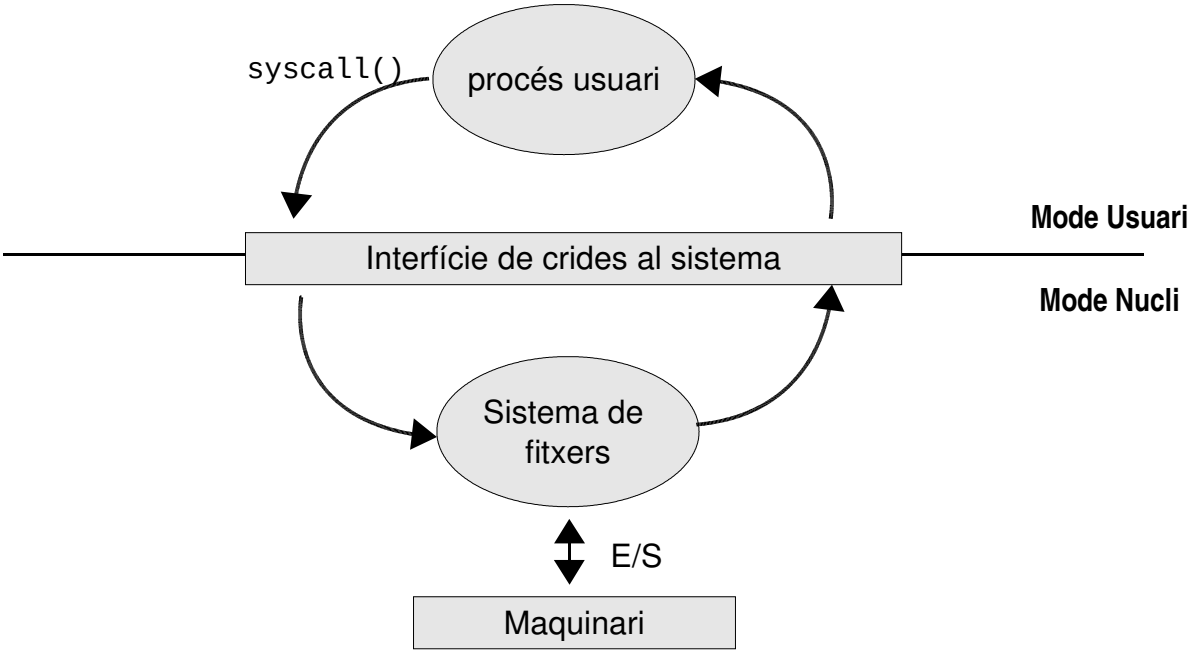
- no es pot accedir directament a cap dispositiu físic: per accedir-hi, haurem de demanar-li al nucli, mitjançant una crida al sistema.
- no es pot accedir a espais de memòria no compartida que pertanyin a un altre procés
- no es té accés a les estructures del nucli (taules de processos, de memòria, ...)

Com que la majoria d'aplicacions que es fan servir habitualment en un ordinador funcionen en mode usuari, aquestes restriccions proporcionen una major estabilitat al sistema. Quan un procés llançat per un usuari qualsevol falla, no hi ha cap repercussió

per la resta de processos que puguin estar funcionant en aquell mateix moment, ja que cada procés utilitza el seu propi espai de memòria i no podrà gastar-ne de la resta de processos actius.

L'accés per part d'un usuari qualsevol a l'espai de nucli es farà sempre mitjançant una interfície de crides al sistema, que decidirà si l'usuari pot accedir o no al recurs sol·licitat.

Aquest és l'esquema[1] d'un exemple de crida al sistema:



1 extret de http://www.e-ghost.deusto.es/docs/Linux_2.6_SysCalls.hack04ndalus.pdf

● Els mòduls del kernel:

El nucli Linux és monolític. Això vol dir que consta d'un únic executable que s'encarrega de realitzar les tasques atribuïbles a un sistema operatiu (gestió de memòria, comunicació entre processos, planificador, sistema de fitxers, entrada/sortida, etc..).

A diferència dels nuclis monolítics, els *micro-kernels* deleguen la realització d'aquestes tasques en un conjunt de funcions o processos que es comuniquen entre ells i amb el nucli mitjançant el pas de missatges. D'aquesta manera, alguns d'aquest processos poden córrer fora del nucli, facilitant un disseny per capes i una programació més modularitzada. Teòricament, els micro-kernels tenen avantatges sobre els nuclis monolítics, però aquests darrers tenen un millor rendiment, segons sembla haver demostrat el pas dels anys i, també, lògicament, gràcies a les millores que s'hi han introduït paulatinament.

Deixant de banda una vella polèmica[1], Linux aprofita els avantatges dels *micro-kernels* gràcies als mòduls. Un **mòdul** és un fitxer objecte que pot ser carregat i descarregat de manera dinàmica per tal que el nucli pugui aprofitar-se de les funcionalitats que implementa. Aquestes funcionalitats poden ser des de la implementació d'un sistema de fitxers o d'un controlador de fitxers, fins a un conjunt d'estructures que ens permetin definir determinades polítiques de seguretat, passant per funcions de xarxa, so i control d'energia, per posar alguns exemples.

A diferència de les capes dels micro-kernels, els mòduls corren a l'espai del nucli, com si es tractessin d'una funció del nucli enllaçada estàticament. D'aquesta manera ens estalviem el pas de missatges (i la seva gestió), millorant el rendiment. D'altres avantages de l'ús de mòduls són:

- Possibilitat de carregar-los i descarregar-los dinàmicament, fet que permet usar els mòduls quan ens convingui, i alliberar els recursos que consumeixen quan no els necessitem.

- Són independents de la plataforma. Els mòduls poden ser desenvolupats de manera que no importi sobre quin maquinari estan corrent, com passa amb la majoria dels mòduls anomenats genèrics.

- Poc consum de recursos. Un cop enllaçats al nucli, la comunicació entre els mòduls i aquest darrer, no necessita del pas de missatges, ja que el codi objecte del mòdul és accessible de manera estàtica pel nucli.

1 http://www.dina.dk/~abraham/Linus_vs_Tanenbaum.html

Per totes aquestes característiques els mòduls del kernel han estat un dels mètodes escollits per implementar polítiques de seguretat als nuclis Linux, ja que entren directament a l'espai d'usuari, on poden operar amb tota llibertat. A més, la possibilitat de carregar i descarregar els mòduls permet activar o desactivar funcionalitats de seguretat de forma dinàmica.

- **Permisos per defecte als sistemes GNU/Linux:**

L'accés al sistema de fitxers ve determinat per tres categories de permisos: *user*, *group* i *others*. Tot fitxer o directori té exactament un propietari, està associat a qualsevol membre d'un grup i la resta d'usuaris podran fer més o menys coses amb el fitxer, segons els permisos que tinguin assignats.

La següent taula resumeix els permisos sobre fitxers:

símbol	*permís*	*descripció*
`r`	lectura	pot obrir-se i llegir-se'n els continguts
`w`	escriptura	pot ser modificat (però no eliminat)
`x`	execució	pot executar-se
`s`	`setuid`/`setgid`	permisos `setuid` o `setgid`
`-`	accés denegat	no pot ser llegit, escrit ni executat, segons la posició del '-'

Els permisos de lectura, escriptura i execució són prou clars i coneguts com per que faci falta explicar-los. Però el que si que explicarem són el permisos de `setuid` i `setgid`. El tractament d'aquests permisos és diferent segons s'apliquin a un fitxer o a un directori.

Activant el bit `setuid` al propietari d'un fitxer i concedint permisos d'execució sobre el mateix, cedim, durant l'execució del programa, la identitat del propietari a l'usuari que l'executi. De manera que l'execució d'aquest fitxer o programa es fa amb els permisos del propietari, com si fos aquest darrer qui l'executa. Això és especialment perillós quan l'usuari `root` activa el bit `setuid` per a que es puguin executar determinats programes, com ara `passwd(1)` o `traceroute(8)`, com a exemples típics. Si l'aplicació no controla correctament tots els aspectes referents a la protecció del sistema, podria provocar-se un accés a privilegis que no hauria de disposar qui ha llançat l'aplicació, i la seguretat del sistema quedaria compromesa.

El comportament del bit `setgid` sobre un fitxer és el mateix que el del `setuid`, però ara parlem de la cessió de la identitat del grup. L'usuari que accedeixi a aquest fitxer disposarà de tots els privilegis atribuïts al grup del fitxer. Com en el cas del `setuid`, és necessari que el fitxer sigui executable per a que funcioni.

Pel que fa als directoris, resumim els permisos amb la següent taula:

símbol	*permís*	*descripció*
`r`	lectura	permet llegir i obrir el directori
`w`	escriptura	poden afegir-se o esborrar-se fitxers al directori
`x`	execució	permet accedir al contingut del directori
`s`	`setgid`	permís de `setgid` (només al grup)
`t`	mode especial	*sticky bit*
`-`	accés denegat	no pot ser llegit, escrit ni executat, segons la posició del '-'

El permís de lectura sobre un directori indica que es pot obrir, però no necessàriament que puguem llegir-ne el contingut. Per poder-ho fer caldrà disposar del permís d'execució sobre el directori, que ens permet accedir al contingut d'un directori i, a més, disposar dels permisos suficients sobre els fitxers que volem llegir.

El permís d'escriptura ens permetrà afegir, eliminar o esborrar fitxers al mateix directori, independentment dels permisos que s'hagin assignat sobre aquests fitxers. Cal destacar que podem eliminar fitxers sense necessitat de disposar del permís d'escriptura sobre el mateix.

L'*sticky bit*, bit enganxifós o *save text bit*, actualment és una opció només vàlida per directoris, tot i que en un origen va ser desenvolupat per a fitxers executables (sol·licitava als sistema operatiu que mantigués en memòria el programa per a futurs usos). Quan està activat per a un directori qualsevol, un usuari només podrà eliminar o modificar els fitxers i directoris que siguin de la seva propietat. L'exemple típic d'un directori marcat amb l'*sticky bit* és `/tmp`, on tothom pot escriure-hi, però només podem eliminar aquells fitxers que haguem creat nosaltres mateixos.

Per acabar, el bit `setgid` permet fixar el grup de tots els fitxers d'un directori. Així, si marquem un directori amb el bit `setgid`, tots els fitxers que es creïn de nou pertanyeran al mateix grup que el del directori que els conté. Aquesta opció va molt bé quan s'ha de treballar en directoris compartits i volem que diferents usuaris pertanyents a un mateix grup puguin treballar sobre els fitxers sense problemes de permisos. Establint un directori amb l'*sticky bit* i permisos d'escriptura per al grup, permetrem que tots els membres del grup puguin escriure i modificar el contingut del directori.

- **Matrius d'accés:**

Hem vist dos dels mecanismes bàsics de seguretat als nuclis de Linux que provenen del món UNIX, la separació de dos entorns d'execució o dominis de protecció (espai usuari i espai nucli) i el sistema de permisos sobre fitxers i directoris.

Amb aquests dos mecanismes, es pot establir una relació entre els dominis de protecció i els objectes del sistema, de manera que podem establir, mitjançant matrius de protecció, quines operacions poden fer-se des d'un domini sobre un objecte del sistema. Aquestes matrius de protecció contenen, a les seves cel·les, els drets d'accés sobre una parella (domini, objecte), o un valor buit, si aquesta parella no té drets assignats.

Per motius d'eficiència, la implementació d'aquestes matrius sol fer-se mitjançant estructures dinàmiques de dades d'una dimensió, i la representació de la matriu per columnes o per files ens donaran dos conceptes diferents: les ACL i les *capabilities*.

En el cas de les ACL (*Access Control List*), la representació de la matriu es fa per columnes, amb una llista per objecte que indica les operacions que pot fer cada domini sobre aquest objecte.

Amb les *capabilities* o capacitats la representació es fa per files, amb una llista per domini que especifica les operacions que poden fer-se sobre un objecte quan es pertany a aquest domini. Aquestes llistes o conjunt de descriptors indiquen a un procés si està autoritzat a realitzar una operació sobre un objecte concret.

Per exemple, un descriptor de fitxer és una *capability* o capacitat. En el moment d'obrir un fitxer, es fa sol·licitant-lo amb uns permisos que, si són vàlids, s'emmagatzemaran en estructures de dades del nucli. En futures operacions de lectura o escriptura, el nucli utilitzarà aquest descriptor i les estructures de dades creades per comprovar els permisos fent una simple consulta a la taula apropiada.

El sistema operatiu proporciona mètodes d'accés a les capacitats i ens permet realitzar unes operacions elementals, que són: copiar capacitat, transferir capacitat entre processos, modificar una capacitat i destruir una capacitat.

- **DAC vs MAC:**

Que un sistema operatiu disposi d'un usuari amb la pràctica totalitat dels permisos, com ho és l'usuari `root` dels sistemes UNIX, ens indica que no és segur. A més, és un inconvenient a l'hora d'administrar-lo, ja que ens imposa una jerarquia molt rígida que no permet distribuir les tasques administratives. Els *Discretionary Access Control* (DAC) segueixen precisament aquest model, gràcies al qual un usuari administrador (i totpoderós) resta exempt del control d'accés als recursos del sistema.

El criteri del DAC és el de la seguretat per propietat. Si un usuari és propietari d'un fitxer, només ell (a part del `root`, és clar) podrà canviar els permisos de lectura, escriptura i execució. No podem establir polítiques d'accés que no estiguin en funció dels usuaris existents al sistema.

Per a poder considerar un sistema segur, doncs, entre d'altres aspectes, hem de garantir un control d'accés sense distincions. Això és el que ens proporciona el model *Mandatory Access Control* (MAC), un control d'accés obligatori per tothom, establert de manera independent al propietari del recurs. Amb el model MAC les polítiques de seguretat afecten a tots els usuaris del sistema, fins i tot al compte `root`, i no poden ser canviades ni esquivades ni tan sols pels propietaris d'aquests recursos. Els models MAC permeten molta flexibilitat i un nivell de refinament (granuralitat) en l'assignació de permisos, que fa que s'adaptin més fàcilment a les necessitats de la vida real.

3.- Què ens aporta de nou el kernel 2.6?

Canvis més importants respecte la versió 2.4

Amb l'aparició de la branca 2.6 del kernel Linux a finals de 2003 (per ser exactes, la seva publicació va ser anunciada el 13 de juliol, tot i que no es va considerar finalitzada fins al desembre de 2003) van ser introduïdes una sèrie de modificacions prou significatives, com per que siguin comentades. Deixant de banda els aspectes referents a la seguretat, que són els que ens ocupen, la nova versió del nucli Linux ha permés que GNU/Linux s'estengui i guanyi terreny en diferents sectors del mercat i ampliï la seva comunitat d'usuaris.

Comentarem a continuació les modificacions més significatives que ha incorporat el kernel 2.6 respecte les branques anteriors, amb noves funcionalitats i parts que han estat reescrites de dalt a baix. Algunes de les modificacions introduïdes ja s'han implementat, incrustades al nucli en alguns casos i com a *patch* en d'altres, a les versions de la branca 2.4.

- **Suport per a més arquitectures:**

Pel que fa a les arquitectures suportades, el nou kernel ha fet grans passos en el camp dels sistemes integrats (*embedded*) i en el de grans servidors de processament paral·lel. En el primer dels casos, el kernel ha desembarcat al mercat dels microcontroladors i ja és present en una gran quantitat d'aparells electrònics com ara agendes electròniques (PDA), sistemes dedicats amb GNU/Linux instal·lat en una memòria flash, telefonia (mòbil i centraletes) i d'altres *gadgets*.

La inclusió de gran part del projecte μClinux[1] al kernel 2.6 ha permès l'acceptació de GNU/Linux en aquest tipus de productes. μClinux va sorgir com a derivat del kernel 2.0 i està destinat a microcontroladors sense unitat de gestió de memòria o MMU, un component indispensable en sistemes multiusuari per protegir la memòria que utiliza cada usuari del sistema, però completament prescindible en sistemes dedicats de baix cost sense funcionalitats d'usuaris.

L'altra direcció que hem comentat és el salt cap a la banda dels grans servidors. Si bé el processament en paral·lel (*Symmetric Multiprocessing* o SMP) ja era suportat des de la

1 http://www.uclinux.org

versió 2.0, la nova branca del kernel inclou suport per a l'arquitectura NUMA (*Non-Uniform Memory Architecture*), el següent pas en multiprocessament després de SMP.

Als sistemes multi-processador els diferents processadors comparteixen el mateix espai de memòria i lluiten per aconseguir-la, ja que només pot accedir-hi un processador a la vegada. Això pot provocar problemes de rendiment que l'arquitectura NUMA mira de solucionar proporcinant un espai de memòria per cada processador.

Per poder acceptar aquesta nova arquitectura al kernel 2.6 s'han retocat algunes parts del kernel i inclòs una nova API que permeti comprendre les relacions entre processadors, memòries assignades i dispositius d'entrada i sortida. A més, el planificador de processos (*scheduler*) ha estat reescrit per que usi un nou algoritme de planificació anomenat O(1), que permet major escalabilitat quan s'incrementa notablement la quantitat de processadors.

Aquestes millores el fan ideal per sistemes que treballin amb multiprocessament, com ho demostra el fet que el kernel 2.6 hagi estat l'escollit per a fer funcionar el supercomputador MareNostrum[1], recentment inaugurat a Barcelona, amb més de 4500 processadors treballant simultàniament.

Altres arquitectures conegudes i suportades a partir de la versió 2.6 del nucli són la de 64 bits d'AMD (x86-64) i la PowerPC 64-bit (ppc64).

També cal destacar les millores en el suport de processadors amb *hyperthreading*, una funcionalitat ja existent en els kernels 2.4, i que actualment només l'ofereixen alguns processadors Pentium 4. Consteix en la capacitat de tractar, a nivell de maquinari, un únic processador com si en fossin dos de lògics, millorant el rendiment del sistema.

Per últim, la possibilitat de disposar d'un kernel interrumpible (*preemptible*) és una de les millores significatives del kernel 2.6. D'aquesta manera es redueix la latència del kernel en events a temps real, ja que es permet aturar processos a baix nivell, encara que ens trobem enmig d'una crida al sistema. Es recomana el seu ús en sistemes d'escriptori, sistemes integrats i sistemes a temps real.

- **Millores en l'escalabilitat:**

A part de la inclusó d'arquitectures que permeten créixer al sistema d'una manera escalable, també s'han modificat els límits interns com poden ser el número d'usuaris, passant el màxim permès de 65000 a més de quatre milions, o el nombre d'identificadors de processos permesos, que ha passat de 32000 a mil milions. Pel que fa a l'emmagatzematge de dades, es dóna suport de 64 bits als dispositius de bloc que el

1 http://www-1.ibm.com/servers/eserver/linux/power/marenostrum/about.html

suportin, fins i tot en plataformes de 32 bits, de manera que es podran arribar a emmagatzemar fins a 16TB en una màquina normal i corrent.

Pel que fa als dispositius, no només se'n suporten més, sinó que també se'n poden tenir més de connectats. S'han augmentat els límits dels tipus de dispositius que podem utilitzar de 255 a 4095, i el nombre d'aparells d'aquest tipus que es poden connectar ha passat de 255 a un milió.

- **Millores en el treball amb els dispositius:**

A mesura que la tecnologia avança, van apareixent nous dispositius i nous sistemes de comunicació entre ells o entre aquests i el nucli. A més, amb el temps, més fabricants han fet públics els seus controladors fent possible la seva integració al sistema GNU/Linux. Tot plegat ha conduït a un major nombre de dispositius amb els quals el kernel és capaç de treballar. S'ha afegit una base de dades de dispositius.

La incorporació de *Plug-and-Play* (PnP) a nivell de BIOS, juntament amb el suport per a velles ISAPnP, les millores en el sistema PCI com ara acceptació de PCI en calent, gestió d'energia i suport per a múltiples targetes AGP, han fet que GNU/Linux sigui un sistema operatiu completament PnP.

Pel que fa a dispositius externs, són remarcables el suport del protocol de comunicació USB 2.0 i les millores introduïdes en els protocols sense fils Bluetooth i en el d'infrarojos IrDA.

Les implementacions dels estàndards IDE i SCSI també han estat modificades per tal de proporcionar major escalabilitat i millorar el suport de dispositius d'emmagatzematge. L'eliminació de les de dependències amb certs mòduls d'SCSI per a poder gravar CD amb gravadores IDE ha estat una de les millores més agraïdes per la comunitat d'usuaris. El protocol Serial-Ata també és suportat al nucli 2.6.

Un altre aspecte destacable són les millores en el model de dispositius, que es busca que sigui el més unificat possible. El carregador de mòduls treballa ara amb fitxers `.ko` (*kernel object*) en comptes de `.o`, mentre que s'han afegit noves funcionalitats en la càrrega i descàrrega de moduls. Les millores afecten als sistema de gestió d'energia dels dispositius(s'ha inclòs ACPI -*Advanced Configuration and Power Interface*- al nucli) i a la comunicació entre el nucli i els dispositius de manera que aquest sap en tot moment: quins aparells hi ha connectats al sistema i quin bus utilitzen, l'estat d'energia per a un dipositiu determinat, quins són els controladors de cada dispositiu i l'estructura de busos del sistema (com es connecten els busos entre ells i quins dispositius estan connectats amb cadascun dels busos) entre d'altres. Com que el nucli és qui passa a tenir tota la informació del maquinari és possible realitzar tasques de detecció de *hardware* i

connectar dispositius en calent amb més facilitat. També s'han afegit noves API que permeten una migració de mòduls de la branca 2.4 a la 2.6 amb major facilitat. A més a l'espai d'usuari s'ha afegit un nou sistema de fitxers anomenat `sysfs` (similar als sistemes `proc` i `devfs`), muntat normalment a `/sys` i que ens dóna una representació del model de dispositius que està utilitzant el nucli en tot moment.

- **Sistemes de fitxers:**

El llistat de sistemes de fitxers suportats pel kernel 2.6 és el següent: `ext2`, `ext3`, `reiserfs`, `jfs`, `xfs`, `minix`, `romfs`, `iso9660`, `udf`, `msdos`, `vfat`, `ntfs` (només lectura), `adfs`, `amiga ffs`, `apple macintosh hfs`, `BeOS befs` (només lectura), `bfs`, `efs` (només lectura), `cramfs`, `free vxfs`, `os/2 hpfs`, `qnx4fs`, `sysvfs` i `ufs`.

A més, els sistemes de fitxers més utilitzats a GNU/Linux (ext2 i ext3 principalment) han estat modificats per a suportar atributs extesos sobre els fitxers, fet que permet implementar les POSIX ACL[1], incrementant la seguretat en el control d'accés, ja que permet l'aplicació d'un model MAC. ACL permet autoritzar o denegar l'accés a un fitxer al propietari, membres del grup o a la resta d'usuaris, al marge dels permisos que tingui assignat el fitxer. Mitjançant matrius d'accés, es defineixen els drets d'accés per a cada objecte del sistema, de manera que la seguretat per propietat deixa de tenir vigència i podem dur a terme un control d'accés més refinat.

- **Aspectes relacionats amb la xarxa:**

La xarxa sempre ha estat un dels punts forts dels sistemes *NIX, i GNU/Linux no és una excepció. Entre les millores aportades pel nou kernel destaca la inclusió de la versió 4 del sistema de fitxers de xarxa NFS (*Network File System*) que aporta un munt de noves funcionalitat a aquest protocol tant estès. Les millores van des de mecanismes d'autenticació més segurs fins a la classificació dels fitxers i del sistema de fitxers per facilitar les implementacions en els servidors NFS.

En el terreny dels protocols de comunicacions, el suport per a IPSec[2] i IPComp[3] ha estat afegit al nucli. El primer, IP Security, proporciona mètodes d'autenticació i xifratge a les comunicacions per xarxa tant a nivell local com via Internet. Concretament, el kernel 2.6 dóna suport a dos mecanismes IPSec: *Authentication Header* (AH) i *Encapsulated Security Payload* (ESP). AH proporciona autentificació a nivell de paquet IP que garanteix que el paquet no ha estat modificat des del punt d'origen fins al receptor, mentre que ESP

1 http://www.suse.de/~agruen/acl/linux-acls/online/
2 http://www.faqs.org/rfcs/rfc2401.html
3 http://www.faqs.org/rfcs/rfc3173.html

proporciona tant xifratge com antentificació a nivell de paquet.

IPComp ve d'IP *Payload Compression* i és un protocol que comprimeix els datagrames IP. Usat juntament amb IPSec permet reduir el tamany dels paquets IP que han estat engrandits per afegir-hi les capçaleres addicionals que requereix IPSec.

Pel que fa a IPv6, el suport per a fer *tunneling* (mecanisme que permet encapsular un protocol o una sessió dins d'un altre) sobre aquest procol s'ha afegit al nucli 2.6, que a la vegada també proveeix les IPv6 *Privacy Extensions*[1]. Aquestes extensions estan orientades a proporcionar anonimat a Internet, permetent als usuaris protegir la seva identitat quan fan servir adreces IPv6.

La inclusó del nou protocol de transport SCTP (*Stream Control Transmission Protocol* [2]) és una altra de les novetats. SCTP garanteix la majoria de les funcionalitats de TCP, però n'aporta d'altres que són força interessants per a determinats serveis com ara la telefonia sobre IP. Entre aquestes característiques destaquen el *multi-streaming*, que permet particionar les dades a transmetre en un conjunt de fluxes (*streams*) de manera que, en cas de produir-se un error, només afecta a un fluxe en concret, i el *multi-homing* que és la capacitat d'oferir múltiples adreces IP des d'una de les bandes de la connexió SCTP, que normalment sol ser el servidor.

- **Àudio i multimèdia:**

La *Advanced Linux Sound Architecture* o ALSA[3] passa a substituir l'antic *Open Sound System* (OSS), tot i que s'ha mantingut la compatibilitat amb OSS. ALSA proveeix de funcionalitats d'àudio i de MIDI a GNU/Linux i dóna suport per a tota mena de dispositius d'àudio, des de targetes de so per a ús domèstic fins a targetes multicanal per a professionals.

Pel que fa al sistema de vídeo, Video4Linux[4] ha estat retocat i millorat per a donar suport a targetes sintonitzadores de televisió i a capturadores de vídeo. Ha aparegut una nova API de Video4Linux (`v4l2`), la segona generació d'aquesta API que resol *bugs* de disseny de la primera versió i que ha estat afegida al nucli.

Finalment, també s'ha afegit el suport de maquinari per a poder fer *Digital Video Broadcasting* o DVB[5].

1 http://www.faqs.org/rfcs/rfc3041.html
2 http://www.faqs.org/rfcs/rfc2960.html
3 http://www.alsa-project.org
4 http://linux.bytesex.org/v4l2
5 http://www.linuxtv.org

- **Sistema de construcció del kernel:**

L'estructuració de la informació de configuració del nucli mostrada quan després d'un `make menuconfig|xconfig` a `/usr/src/linux` ha millorat considerablement al kernel 2.6. Hi ha molta més coherència i els ítems estan agrupats d'una forma més ordenada, fet que l'usuari final agraeix notablement.

Però a banda d'aquests canvis més estètics, el procés de compilació del nucli també ha canviat. Alguns *flags* clàssics han deixat d'existir (ja no cal fer `make dep`), s'han afegit eines gràfiques de configuració (`make xconfig` per llibreries Qt i `make gconfig` per les GTK) i tot el procés és, en general, més ràpid.

- **Criptografia i seguretat:**

Una API genèrica de criptografia ha estat afegida al nucli, donant suport a diversos del algoritmes de xifrat més coneguts com ara MD5, SHA-1, DES, Triple DES, Blowfish i AES entre d'altres (n'hi ha més de 20 de suportats). Algunes de les noves funcionalitats del kernel, com ara IPSec, requereixen d'aquesta API per a poder funcionar correctament.

Pel que fa als aspectes de seguretat, que són els que centren aquest treball, i que comentarem amb més extensió als propers apartats, cal destacar la modularització de la seguretat al kernel, amb la inclusió d'una nova secció a les opcions de compilació del nucli anomenada *Security Options*. Aquesta secció ens proporciona diferents models de seguretat que podem aplicar al nostre sistema, i que són:

- Ganxos de seguretat per sockets i per la xarxa.
- Linux Capabilities
- Suport per a *Root Plug*
- Nivells de seguretat BSD
- Suport per a NSA SELinux

Altres millores com impedir l'exportació de la taula de crides al sistema, evitant o dificultant[1] la suplantació de les mateixes des d'un mòdul de kernel, o la possibilitat de fer funcionar UML (*User Mode Linux*[2]), un mecanisme que permet fer córrer una o més màquines Linux virtual dins de l'espai d'usuari amb el seu espai de memòria propi i el seu directori arrel. Se sol utilitzar per engabiar aplicacions o fer experiments sense que la màquina real resulti malmesa.

Tot plegat, a banda de les millores de seguretat breument comentades als apartats de sistemes de fitxers i de xarxa, demostren l'interès que els desenvolupadors del nucli han posat en els aspectes relacionats amb la seguretat.

1 http://www.e-ghost.deusto.es/docs/Linux_2.6_SysCalls.hack04ndalus.pdf
2 http://user-mode-linux.sourceforge.net

4.- Mòduls de seguretat incorporats al kernel 2.6

Una de les millores més remarcables en qüestions de seguretat als sistemes GNU/Linux amb la versió 2.6 del nucli, és la modularització de les tasques de seguretat al sistema, propocionant una interfície que permet crear mòduls carregables al kernel, tot i que es manté la possibilitat de compilar els mòduls al kernel. Fins les versions anteriors, quan es volien afegir funcionalitats al kernel s'havia de fer mitjançant *patchs* de seguretat, que modifiquen i/o amplien el codi del kernel per tal de cobrir mancances de seguretat. L'inconvenient dels *patchs* és que són compilats de manera estàtica i no podem carregar-los ni descarregar-los segons ens convingui.

Per tal d'assolir l'objectiu de permetre la inclusió de nous mòduls de seguretat, va aparèixer el projecte Linux Security Modules (LSM), que en els seus orígens també va ser implementat com a *patch* i que s'ha integrat com a opció de configuració al kernel a la branca 2.6. En aquesta secció farem una introducció al nou entorn LSM així com a la seva implementació. També comentarem els diferents models de seguretat que s'han implementat en forma de mòduls i que es distribueixen amb les versions actuals del nucli. D'aquest mòduls n'hi ha de realment potents, com ara les Linux Capabilities o l'NSA SELinux i d'altres que s'han afegit al kernel com a exemple o per cobrir algunes noves funcionalitats. Amb el pas del temps segurament s'aniran afegint nous mòduls al nucli que permetran cobrir un ventall més ampli de models de seguretat.

Per motius d'organització del document i degut a la llargària de les seccions, hem separat l'explicació dels mòduls Linux Capabilities i NSA SELinux en apartats independents. Així, en aquesta primera secció sobre els mòduls de seguretat veurem el model LSM així com la seva implemetació i tres dels models de seguretat incorporats a al kernel Linux 2.6. Linux Capabilitites i NSA SELinux els veurem a les seccions 5 i 6, respectivament.

4.1.- Linux Security Modules (LSM):

El projecte Linux Security Modules (LSM) va néixer amb la idea de proporcionar un f*ramework* de propòsit general per al control d'accés que fos ràpid i lleuger. Una bona manera de millorar la seguretat als *hosts* és afegir models de control d'accés al kernel, ja que ens proporcionen un model de tipus MAC que hem vist als conceptes previs. A banda d'això, la possibilitat implementar modes de control d'accés en forma de mòduls carregables ha estat una altra de les contribucions del projecte LSM

A continuació, farem una ullada als aspectes més bàsics dels LSM i també veurem com s'implementen dins del nucli.

• Introducció:

L'aparició, el març de 2001, de la distribució *Security-Enhanced Linux* (SELinux), desenvolupada per la NSA (*National Security Agency*) nord-americana, que proporcionava un control d'accés de tipus MAC, juntament amb l'existència d'altres projectes de seguretat que es podien aplicar al nucli en forma de patch, va fer prendre consciència de la necessitat d'avançar en el camp de la seguretat al kernel.

Com a resposta a aquestes necessitats i projectes, Linus Torvalds va realitzar un conjunt de d'observacions per tal descriure les consideracions que s'haurien de tenir en compte a l'hora d'incloure una estructura de seguretat (*security framework*) dintre del codi principal del nucli Linux. Va descriure un *framework* general que hauria de proporcionar un conjunt de ganxos de seguretat (*security hooks*) per tal de controlar les operacions sobre els objectes del kernel i un conjunt de camps de seguretat (*security fields*) a les estructures de dades del kernel per tal de mantenir certs atributs de seguretat. Aquest *framework* s'hauria de poder usar mitjançant mòduls del kernel que permetessin implementar qualsevol model de seguretat.

El projecte Linux Security Modules (LSM) va ser ideat per a desenvolupar aquesta estructura de seguretat, i començat per l'empresa WireX Communications el mateix 2001 (a l'actualitat absorbida per Novell[1]). LSM va ser el resultat de la unió de diferents projectes de seguretat com ara Immunix (actualment també treballen per Novell), SELinux i Janus[2] entre d'altres, juntament amb desenvolupadors individuals. L'objectiu de LSM era desenvolupar un *patch* del kernel que implementés el *framework* desitjat.

Així doncs, la idea era desenvolupar un *patch* que permetés la inclusió de mòduls de seguretat al kernel de Linux i més concretament o, almenys, als inicis, la inclusió de mòduls de control d'accés. La nova estructura en si, però, no proporciona cap mena seguretat, sinó que proveeix de la infraestructura que permetrà donar suport a aquests mòduls de seguretat.

A grans trets, seguint les consideracions fetes per Linus Torvalds, LSM afegeix un conjunt de camps de seguretat (*security fields*) a les estructures del kernel i incrusta, als punts crítics del codi del kernel, un seguit de crides a ganxos (*hooks*) que permeten gestionar els camps de seguretat i dur a terme un control d'accés. A més, també proveeix d'un conjunt de funcions que permeten la càrrega o registre i descàrrega de mòduls de seguretat

1 http://www.novell.com
2 http://www.cs.berkeley.edu/~daw/janus/

Els camps de seguretat estan a dins de les estructures de dades (`struct`) que permeten controlar les diferents tasques que realitza el kernel. Així, hi ha camps de seguretat per operacions sobre sistemes de fitxers, operacions d'execució de processos i programes, operacions amb *pipes*, *sockets*, inodes i fitxers, per als dispositius de xarxa i per operacions relacionades amb el System V IPC de comunicació de processos (semàfors, cues de missatges, gestió de memòria). La implementació d'aquests *security fields* es fa mitjançant punters a `(void *)`.

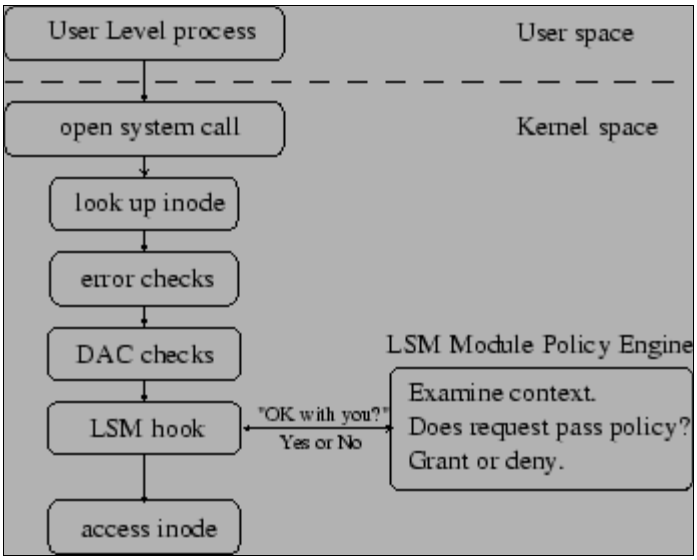
Pel que fa als ganxos, són punters a funcions contingudes en una taula global anomenada `security_ops`, que és del tipus de dades `security_operations`, definit al fitxer de capçalera `<include/linux/security.h>`. L'estructura `security_operations` conté més de 130 ganxos relacionats amb els objectes del kernel que permeten gestionar la seguretat de les operacions del sistema.

Tot i que els ganxos LSM estan organitzats en estructures basades en els objectes del kernel, poden distingir-se dos tipus principals de *hooks*: els que gestionen els camps de seguretat i els que s'usen per dur a terme un control d'accés. Amb l'ajuda de `grep(8)`, buscant per "`security_ops->`", podrem localitzar les crides als *hooks* dins del codi del kernel.

La taula global `security_ops` s'inicialitza amb un conjunt de ganxos proveïts per un mòdul de seguretat primari. Mitjançant les funcions definides a l'anterior fitxer de capçalera i anomenades `register_security`, `unregister_security`, `mod_reg_security` i `mod_unreg_security` és possible accedir a la taula `security_ops` per tal d'activar o anul·lar els nostres propis *hooks*. Les dues primeres funcions permeten, com el seu nom indica, registrar o treure els ganxos de la taula, i les dues últimes permeten invocar aquest ganxos en el moment que més convigui, deixant tota la llibertat per als desenvolupadors de possibles mòduls de seguretat.

L'abstracció bàsica de la interfície LSM és la mediació entre els events a l'espai d'usuari i els objectes interns del kernel. A partir de les crides al sistema i baixant fins arribar als objectes del kernel, el *hooks* es col·loquen just abans d'accedir als objectes per permetre comprovar si qui hi està accedint pot realitzar l'operació que vol fer sobre l'objecte del kernel en qüestió.

La següent figura[1] representa el procés:



Per últim, i com a nota anecdòtica, en alguns documents del projecte LSM es parla de la possibilitat (i fins i tot de l'existència) d'una nova crida al sistema, anomenada `security`, implementada pel propi projecte. A les versions actuals del kernel, aquesta funció encara no ha estat implementada, i a la documentació sobre LSM proporcionada amb el codi font del nucli ja no apareix aquesta crida per enlloc. Tot i que la intenció d'aquesta nova crida al sistema era facilitar la gestió dels mòduls de seguretat, la dificultat a l'hora d'introduir-la al codi principal del kernel així com la repercussió que aquestes modificacions podrien tenir sobre algunes aplicacions d'usuari molt esteses, poden haver estat motius per la seva no implementació.

1 extreta de http://lsm.immunix.org/docs/lsm-usenix-2002/html/node3.html

• Implementació dels LSM:

Podem considerar com a quatre les principals modificacions realitzades sobre el kernel per tal d'integrar-hi el projecte LSM:

- Addició d'un seguit de camps de seguretat dins d'algunes de les estructures de dades del kernel.
- Inserció, en punts clau del codi del kernel, de les crides a les funcions referenciades pels *security hooks*.
- Creació de noves funcions destinades a permetre als mòduls de seguretat registrar-se o desvincular-se del nucli.
- Migració del sistema de *capabilities*, integrat en la seva major part al kernel, cap a un mòdul de seguretat carregable.

A continuació, passem a descriure cadascuna d'aquestes novetats amb una mica més de detall.

• Camps de seguretat:

Com s'ha dit a la introducció, els *security fields* són punters a `(void *)` que permeten als mòduls de seguretat associar informació de seguretat en alguns dels objectes del kernel. Aquests objectes solen estar implementat mitjançant estructures de dades del nucli, que és on s'han afegit els camps de seguretat.

Les estructures que han estat modificades, així com l'objecte abstracte que representen, apareixen a la següent taula[1]:

1 extreta de http://lsm.immunix.org/docs/lsm-usenix-2002/html/node6.html

estructura de dades	objecte del kernel
`task_struct`	tasca o procés
`linux_binprm`	programa
`super_block`	sistema de fitxers
`inode`	fitxer, tubs (*pipes*) o sockets
`file`	fitxer obert
`sk_buff`	buffer de xarxa
`net_device`	dispositiu de xarxa
`kern_ipc_perm`	semàfors, gestió de memòria compartida i cues de missatges
`msg_msg`	missatges individuals

Aquestes estructures estan definides en diferents fitxers del codi, i es poden buscar amb una mica de paciència pels directoris que contenen les fonts. Així, per posar un parell d'exemples, la `task_struct` està definida a `<include/linux/sched.h>` i la `inode` ho està a `<include/linux/fs.h>`. Consultant aquests fitxers i els que defineixen la resta d'estructures podrem trobar-hi els camps de seguretat.

Un cop tenim els camps de seguretat definits, la seva activació i la gestió de la informació de seguretat que puguin contenir recau sobre el mòduls de seguretat. La interfície LSM simplement proporciona aquests camps i un seguit de crides als *security hooks* que poden ser implementades segons les necessitats de cadascú. Per a la majoria dels objectes del kernel hi han definits dos ganxos, l'`alloc_security` i el `free_security`, que permeten reservar memòria per les estructures de dades que contenen la informació de seguretat, així com alliberar l'espai reservat. De la mateixa manera, també es proporciona un ganxo que permet actualitzar la informació de seguretat quan faci falta.

Degut al fet que hi ha objectes del kernel que es creen abans que la càrrega dels mòduls de seguretat sigui efectiva, el control d'aquest tipus d'objectes ha de ser dut pels mòduls de seguretat. Per fer-ho, el projecte LSM proposa tres aproximacions: en primer lloc existeix la possibilitat de, simplement, ignorar aquesta mena d'objectes, tractant-los com si fossin elements externs del mòdul i deixant que siguin controlats únicament pel sistema de control d'accés establert per defecte a Linux (per norma general, les *capabilities*). Una segona possibilitat és activar els camps de seguretat d'aquests objectes previs durant la inicialització del mòdul de seguretat, recorrent el codi del kernel per tal d'activar-los un a un, tot tractant els problemes de sincronisme que poden produir-se durant el procés. Per últim, podem fer la comprovació de si un objecte té un *security field* activat quan el necessitem, i activar-lo si ens fes falta.

- Crides a les funcions de seguretat a través dels ganxos:

Com ja s'ha comentat a la introducció, cada ganxo LSM és un punter a una funció d'una taula global anomenada `security_ops`. Aquesta taula, del tipus de dades `struct security_options`, consisteix en un conjunt de subestructures que agrupa els ganxos relacionats amb determinats objectes del kernel, vistos a la última taula.

Malgrat aquesta organització dels ganxos, podem establir dues categories principals de *hooks*: els que s'utilitzen per gestionar (reserva i alliberament d'espai de memòria i activació) els *security fields* i els que s'utilitzen per dur a terme un control d'accés. Els ganxos `alloc_security` i `free_security` que hem vist fa un moment formen part del primer grup, mentre que el ganxo `inode_permission`, que comprova els permisos d'accés quan s'accedeix a un inode, pertany a la segona categoria.

Un llistat complet dels ganxos definits per a cada versió del kernel el podem trobar a `<include/linux/security.h>`, on podrem trobar-hi també una mica més d'informació de quina és la funcionalitat que cobreix cadascun d'ells. La implementació de les funcions les podrem trobar al codif font del nucli en un fitxer situat a `<security/security.c>`.

Més endavant veurem amb una mica més de detall els tipus específics de *hooks* que implementa LSM.

- Registrant els mòduls de seguretat:

La inicialització del *framework* LSM es realitza durant la seqüència d'arrencada del kernel mitjançant un seguit de ganxos postissos (*dummy hooks*) que reforcen la seguretat tradicional dels sistemes UNIX amb un superusuari i permeten la càrrega d'altres mòduls. Un cop activat el framework, tot dependrà dels mòduls que vulguem utilitzar.

Pel que fa als mòduls de seguretat, durant la seva càrrega s'han d'autoregistrar al *framework* LSM realitzant una crida a la funció `register_security`. Aquesta funció insta a la taula global `security_ops` a tenir una referència cap als punters a funcions que apunten als *hooks* del mòdul, per tal que el kernel el cridi a l'hora de prendre decisions de control d'accés. Cal destacar que en cap cas la funció `register_security` sobreescriurà un mòdul prèviament registrat.

Quan intentem registrar un mòdul i ens falla el procés, probablement és degut a l'existència prèvia d'un altre mòdul de seguretat. En cas que aquest mòdul accepti la càrreg de mòduls secundaris que l'ajuden en la seva feina, disposem de la funció `mod_reg_security` que ens permetrà registrar aquest mòdul secundari. Si amb aquesta funció no podem carregar-lo vol dir que algun mòdul de seguretat preexistent i funcionant

ens ho impedeix, probablement fruit d'una decisió de la política de seguretat.

Que un mòdul de seguretat pugui descarregar-se o deshabilitar-se a si mateix és una decisió del desenvolupador del mòdul i de les necessitats que vulgui cobrir. En cas de poder-ho fer, en el moment de la seva descàrrega, el mòdul haurà d'eliminar les entrades que fan referència a ell a la taula `security_ops`. Amb aquesta finalitat existeix la funció anomenada `unregister_security`, que substitueix els *hooks* del mòdul pels valors per defecte del sistema.

De la mateixa manera que existeix `mod_reg_security`, també disposem de la seva funció equivalent per a la deshabilitació de mòduls secundaris, la `mod_unreg_security`. El fet que un mòdul de seguretat sigui secundari o no serà important a l'hora de deshabilitar mòduls, de manera que si realitzem el registre així (seguint l'exemple proposat en aquest article[1]):

```
/* register ourselves with the security framework */
if (register_security (&rootplug_security_ops)) {
   printk (KERN_INFO
       "Failure registering Root Plug module "
       "with the kernel\n");
   /* try registering with primary module */
   if (mod_reg_security (MY_NAME,
                         &rootplug_security_ops)) {
       printk (KERN_INFO "Failure registering "
              "Root Plug module with primary "
              "security module.\n");
       return -EINVAL;
   }
   secondary = 1;
}
```

Quan vulguem que el mòdul quedi deshabilitat, haurem de comprovar de quin tipus és, de la següent manera:

1 http://www.linuxjournal.com/article/6279

```
/* remove ourselves from the security framework */
if (secondary) {
   if (mod_unreg_security (MY_NAME,
                           &rootplug_security_ops))
      printk (KERN_INFO
             "Failure unregistering Root Plug "
             "module with primary module.\n");
} else {
   if (unregister_security (
       &rootplug_security_ops)) {
      printk (KERN_INFO "Failure unregistering "
              "Root Plug module with the kernel\n");
   }
}
```

Oferint aquesta interfície amb dos nivells de mòduls, els primaris i els secundaris, LSM deixa en mans dels dissenyadors de les polítiques de seguretat si permeten o no l'apilament de mòduls. En cas de permetre-ho, podem fer que sigui el mòdul primari qui s'encarregui de decidir si accepta o no les peticions de `mod_reg_security`.

- Capabilities:

Les Capabilities, definides al document POSIX 1003.1e i que veurem més endavant quan parlem del mòdul LSM que les implementa, permeten dividir el poder del superusuari tradicional dels sistemes *NIX en un seguit de capacitats que pot tenir activades o no. D'aquesta manera s'aconsegueix una mica més de granuralitat en el control d'accés, ja que no només mirem si és `root` o no, sinó que també comprovem si el `root` pot realitzar l'operació que li sol·licita al kernel.

En aquest apartat simplement enumerarem les modificacions que s'han realitzat al kernel, que ja incorpora unes quantes de les *capabilities* POSIX 1003.1e per defecte, per a poder-hi integrar el mòdul LSM i ens hi recrearem una mica més a la secció 5 del document.

El mòdul de Capabiliites del projecte LSM proporciona un *hook* que utilitza les crides a una interfície ja existent anomenada `capable`[1] i que era cridada des d'uns quants punts del codi del kernel per a dur a terme comprovacions de control d'accés mitjançant les *capabiliites*. Així, el mòdul LSM reescriu la funció `capable` per a que cridi a un *hook* LSM i s'aprofita de les crides ja existents al codi del nucli, evitant-se les complicacions d'haver de retocar codi del nucli.

Caldrà doncs, definir definir un *security field* i un ganxo que el gestioni. El *hook* és el punter a la funció `capable`, mentre que el camp de seguretat s'afegeix a l'estructura

1 la podem trobar al fitxer <`security/security.c`>.

`task_struct` definida, com ja s'ha dit, a `<include/linux/sched.h>`. El conjunt de capacitats d'un procés s'emmagatzema en aquesta mateixa estructura en forma de vector de bits, permetent que el camp de seguretat pugui saber en tot moment quines capacitats té activades el procés.

De la mateixa manera, s'han encapsulat les crides al sistema ja existents per gestionar les capacitats, `capset(2)` i `capget(2)`, mitjançant crides a ganxos LSM.

• Tipus de ganxos implementats per LSM:

Hem vist que el projecte LSM ens proporciona una sèrie de ganxos que permeten controlar determinats objectes del kernel, així com la seva implementació i integració dins del codi del nucli. A continuació farem un repàs de quins són els principals tipus de ganxos LSM i quins objectes controlen.

• Ganxos sobre procesos o tasques:

Existeix un conjunt de ganxos sobre els processos que habiliten als mòduls de seguretat per gestionar la informació de seguretat relacionada amb un procés i per dur a terme operacions de control sobre aquest procés. Per mantenir la informació de seguretat d'un procés disposem d'un camp de seguretat a l'estructura de dades `task_struct`, que ens proporciona un control sobre les operacions entre processos, com pugui ser la comunicació entre processos per via de senyals. Aquest camp de seguretat també permet controlar la petició d'operacions privilegiades per part del procés en curs.

• Ganxos per la càrrega de programes:

Molts mòduls de seguretat necessiten la capacitat de realitzar canvis de privilegis quan un procés s'acaba d'executar. Per aconseguir-ho LSM proporciona un seguit de ganxos que són cridats durant l'execució d'una crida a `execve(2)`. Gràcies a dos *hooks* situats a l'estructura `linux_binprm`, definida a `<include/linux/binfmts.h>`, és possible mantenir informació de seguretat durant la càrrega d'un programa. El primer ganxo permet realitzar un control d'accés abans de carregar el programa, mentre que el segon ens deixarà actualitzar la informació de seguretat del procés resultant un cop s'hagi llançat el programa.

- Ganxos pel System V IPC:

Gràcies als ganxos IPC, els mòduls de seguretat poden gestionar informació de seguretat i realitzar un control d'accés sobre el System V IPC. Les estructures de dades dels objectes IPC comparteixen una altra subestructura de dades definida a `<include/linux/ipc.h>` i anomenada `kern_ipc_perm` que conté un *security field* LSM mitjançant el qual es poden fer les comprovacions de permisos. Amb un altre camp de seguretat a l'estructura de dades `msg_msg`, definida a `<include/linux/msg.h>`, LSM permet gestionar la informació de seguretat relacionada amb els missatges individuals.

LSM ha afegit un *hook* a la funció `ipcperms`, implementada a `ipc/util.c` del directori que conté les fonts del kernel, de manera que un mòdul de seguretat pugui utilitzar-lo quan ha de comprovar els permisos del System V IPC. A més a més, LSM també ofereix ganxos cap a les operacions individuals dels objectes IPC que proporcionen informació més detallada sobre el tipus d'operació que es vol realitzar, així com dels arguments rebuts.

- Ganxos pel sistema de fitxers:

Per les operacions sobre fitxers, LSM defineix tres tipus de conjunts de ganxos: sobre el sistema de fitxers, sobre els inodes i sobre els fitxers pròpiament dits. Per fer-ho, hi ha definit un camp de seguretat per cadascuna de les estructures de dades del kernel que implementen aquests objectes: `super_block`, `inode` i `file`, totes tres definides al fitxer de capçalera `<include/linux/fs.h>`.

Els ganxos sobre el sistema de fitxers permeten controlar operacions com ara el muntatge de dispositius de bloc utilitzant `mount(8)` o l'obtenció d'informació del sistema de fitxers amb `statfs(2)`. Pel que fa als ganxos sobre els inodes, LSM s'aprofita de la funció existent `permission`, definida també a `<include/linux/fs.h>`, per inserir-hi un ganxo que ens permetrà realitzar la gestió de seguretat dels inodes, juntament amb un seguit d'altres ganxos que proporcionen més granuralitat sobre les operacions individuals amb inodes. Per últim, els *hooks* sobre els fitxers ens deixaran fer comprovacions extres sobre operacions amb fitxers com són la lectura (`read(2)`) i l'escriptura (`write(2)`) o la validació de permisos que permeten realitzar canvis en la política de seguretat en calent.

- Ganxos per la xarxa:

L'accés a la xarxa per part d'aplicacions corrent en l'espai d'usuari es negocia mitjançant un seguit de ganxos sobre les *sockets*. Aquests ganxos s'interposen a totes les crides al sistema que tenen a veure amb les *sockets*, de manera que tots els protocols basats en aquests objectes poden disposar d'un mecanisme de seguretat gràcies al projecte LSM.

En aquest cas, un *security field* ha estat afegit a l'estructura de baix nivell `sock`, definida al fitxer de capçalera `include/net/sock.h`, dins del directori amb les fonts del kernel. Els ganxos sobre les *sockets* permeten una mediació entre el tràfic de xarxa i els processos que volen utilitzar aquestes dades, ampliant el marc de control d'accés a la xarxa del kernel. A més a més, hi han implementats més ganxos pels protocols IPv4 i Netlink i per les *sockets* del domini Unix.

Les dades que volen viatjar per la xarxa creuen la pila de protocols[1] encapsulades en una estructura anomenada `sk_buff` (*socket buffer*) i definida a `include/linux/skbuff.h` dins del directori amb les fonts del kernel. Afegint un camp de seguretat en aquesta estructura es fa possible gestionar l'estat de la seguretat del paquet de dades mentre creuen les diferents capes de xarxa existents.

Per acabar, els dispositius de xarxa físics i lògics, encapsulats a l'estructura `net_device` definida a `<include/linux/netdevice.h>`, on també ha estat afegit un camp de seguretat que ens permet mantenir i gestionar un estat de seguretat basat en els dispositius.

Existeix un mòdul LSM que veurem de seguida que permet activar tots els ganxos LSM sobre les operacions relacionades amb la xarxa.

- Altres ganxos:

Per acabar amb aquesta secció sobre els tipus de ganxos proporcionats pel projecte LSM queden per comentar dos tipus de *hooks* addicionals: el ganxos sobre els mòduls del kernel i els ganxos relacionats amb operacions de sistema d'alt nivell. Els ganxos sobre els mòduls ens permetran controlar operacions del kernel en el moment de crear, inicialitzar i eliminar mòduls del kernel. Pel que fa als ganxos de les operacions de sistema, ens permetran controlar operacions com ara establir el *hostname* del sistema, accedir a ports d'E/S o la gestió de processos. Malgrat que moltes d'aquestes operacions ja són controlades mitjançant les capabilities, amb els ganxos LSM es permet més granuralitat en el control d'accés.

1 capes d'aplicació, presentació, sessió, transport, xarxa, enllaç i física

4.2.- Models de seguretat inclosos al kernel 2.6:

En aquest apartat veurem quins són els models de seguretat que s'afegeixen a les últimes versions de la branca 2.6 del kernel de Linux. Aquests models han estat implementat en forma de mòduls LSM que hem vist a la secció anterior. Com ja s'ha comentat, dels models proporcionats al nucli destaquen per sobre dels altres els que implementen les Linux Capabilities i la NSA SELinux. Comentarem en aquesta secció els més senzills i deixarem aquests dos, bastant més robustos, per les seccions 5 i 6, respectivament.

• Socket and Network Hooks:

Hem vist a l'apartat anterior el projecte LSM i els mòduls de seguretat que proporciona al kernel per mitjà de camps de seguretat i *hooks* repartits pel codi del nucli Linux. Els *hooks* de xarxa i *sockets* estan definits, com tots els altres, a `<include/linux/security.h>`. Aquests ganxos ens permetran realitzar un control d'accés més exhaustiu (o granular) del tràfic de dades que hi ha entre la xarxa i la nostra màquina. A la secció anterior hem vist com s'implementaven aquest ganxos, i també que afecten a les estructures de dades del kernel que fan referència a les operacions de xarxa: les *sockets* i els seus *buffers*, els dispositius de xarxa (físics i lògics) i alguns dels protocols de xarxa més utilitzats.

Activant aquest mòdul de seguretat podrem realitzar determinades operacions de control sobre la xarxa no habilitades per defecte. Un llistat complet dels ganxos disponibles en activar aquest mòdul el trobarem al fitxer de capçalera esmentat fa un moment, just entre les línies on hi posa:

```
#ifdef CONFIG_SECURITY_NETWORK
```

I que s'acaben amb el seu corresponent

```
#endif
```

Al fitxer de capçalera esmentat hi podem trobar tant els prototipus com les implementacions de les funcions apuntades pels ganxos. Hi ha un total de 20 funcions amb les quals podem controlar tota mena d'operacions relacionades amb les *sockets* (creació, destrucció, escolta, *binding*, enviament i recepció de dades,...) i amb la xarxa en general.

- Instal·lació:

Per activar aquest mòdul i poder treballar amb aquest conjunt de ganxos ho farem a través del menú de configuració del kernel, anant a *Security Options -> Socket and Networking Security Hooks*. A diferència d'altres mòduls LSM, aquest només pot ser compilat directament al kernel i no existeix en una versió carregable. L'opció de configuració del kernel és la `CONFIG_SECURITY_NETWORK`.

- **Root Plug Support:**

Segons es pot llegir a la pròpia secció d'ajuda que apareix a l'hora de seleccionar aquest mòdul al menú de configuració del kernel, el seu ús no es recomana massa ja que, en realitat, es tracta d'un exemple d'un mòdul LSM. El seu desenvolupament va ser realitzat per mostrar com és de senzill crear un mòdul del kernel que utilitza l'entorn de treball LSM.

La funcionalitat pròpia del mòdul és evitar que s'executi qualsevol programa amb `egid = 0` (el de `root`) a no ser que hi hagi connectat a la màquina un cert dispositiu USB, a l'estil de les velles motxilles de protecció de programari propietari.

El codi font d'aquest mòdul pot trobar-se, dins del directori amb les fonts del nucli, a `<security/root_plug.c>`, i és en aquest fitxer on podríem canviar el dispositiu que volem que es comprovi si està connectat (el dispositiu USB que es consulta per defecte és un conversor serial-USB genèric).

A banda de les peculiaritats de la construcció d'un mòdul del kernel, que se'ns escapen dels objectius d'aquest treball, en aquest fitxer podem veure que el cos principal de mòdul (la funció `rootplug_bprm_check_security()`) el que fa bàsicament és, quan detecta que l'`egid` del procés és 0, busca un dispositiu amb un identificador de fabricant i de distribuïdor específic. En cas de no trobar-lo, mostra un missatge per pantalla i avorta l'execució del programa tot retornant un codi d'error de falta de permisos.

En el moment de la càrrega i descàrrega del mòdul es fa ús de les funcions proporcionades per l'entorn de treball LSM per a registrar polítiques i mòduls de seguretat que hem vist amb anterioritat.

- Instal·lació:

L'activació d'aquest mòdul es fa des del menú de configuració del kernel, seleccionant l'opció `CONFIG_SECURITY_ROOTPLUG` que podem trobar a *Security Options -> Root Plug Support*. Un cop compilat, el mòdul s'anomenarà `root_plug` i pot carregar-se amb `insmod(8)`.

• BSD Secure Levels:

Basat en els nivells de seguretat BSD[1], estaven incrustats al codi del nucli a la seva versió 2.0. Amb el temps, les necessitats de seguretat del kernel anaven augmentant, i l'equip de desenvolupadors de les *BSD Secure Level* va utilitzar-ne el codi per escriure el de les Linux Capabilities, a partir de la versió 2.1 del kernel.

Tot i l'existència de similituds entre el funcionament de les Linux Capabilities i els nivells de seguretat BSD, alguns administradors provinents del món BSD enyoraven l'estil BSD i, el 2004[2], un desenvolupador va aportar un mòdul LSM que els implementa.

Hi ha quatre nivells de seguretat BSD que van del més dèbil, el -1, fins al més segur, el 2. Les operacions permeses afecten a tots els usuaris del sistema, superusuari inclòs, i varien segons el nivell de seguretat:

- Nivell -1 : Permanentment insegur
  - No podem d'augmentar el nivell de seguretat.

- Nivell 0 : Insegur
  - No podem fer crides a `ptrace(2)` per al procés init.

- Nivell 1: L'establert per defecte
  - Els dispositius `/dev/mem` i `/dev/kmem` són només de lectura.
  - Els atributs extesos `IMMUTABLE` i `APPEND` d'un fitxer, en cas d'estar activats, no poden ser desactivats.
  - No podem carregar ni descarregar mòduls del kernel.
  - No podem escriure directament a dispositius de blocs muntats.
  - No podem realitzar operacions d'E/S en cru (*raw I/O operations*).
  - No podem realitzar tasques d'administració de la xarxa.
  - No podem activar el bit de `setuid` de cap fitxer.

- Nivell 2: Segur
  - No podem fer retrocedir el rellotge del sistema.
  - No podem escriure a cap dispositiu de bloc, estigui muntat o no.
  - No podem muntar ni desmuntar sistemes de fitxers.
  - No podem rebaixar el nivell de seguretat.

1 http://www.freebsd.org/doc/en_US.ISO8859-1/books/faq/security.html#SECURELEVEL
2 http://lwn.net/Articles/72890

- Instal·lació i ús:

Per instal·lar-nos aquest mòdul al sistema haurem d'activar l'opció de configuració del kernel `CONFIG_SECURITY_SECLVL`, accessible des de *Security Options -> BSD Secure Levels* al menú de configuració i tirar de l'`insmod(8)`. El mòdul en qüestió es diu `seclvl` i també és possible compilar-lo directament dins del nucli.

El nivell de seguretat per defecte quan el mòdul està instal·lat i actiu és el nivell 1, mentre que si l'hem compilat dins del nucli serà el 0. Per canviar aquests valors s'accedeix com a superusuari directament al sistema de fitxers `sysfs` (normalment muntat a `/sys`) per canviar el valor del nivell, així:

```
:~# echo -n "2" > /sys/seclvl/seclvl
```

Una altra manera d'indicar-li al mòdul quin nivell de seguretat volem activar és fent-ho en el moment de la càrrega del mòdul, passant el nivell per paràmetre.

```
:~# insmod /lib/modules/2.6.12.1/kernel/security/seclvl.ko initlvl=2
```

Aquests nivells de seguretat funcionen a nivell del kernel, de manera que no podran ser desactivats seguint els procediments habituals com ara `rmmod(8)` un cop estiguin habilitats. Com en el cas de les Linux Capabilities, caldrà tornar a engegar la màquina per a tornar al nivell per defecte, a no ser que seguim el procés que expliquem a continuació.

Per facilitar la vida al superusuari, en el moment de la càrrega del mòdul, podem especificar un paràmetre (`sha1_passwd`) amb una clau xifrada amb l'algorisme SHA1 que ens permetrà rebaixar el nivell de seguretat al 0, on podrem desactivar el mòdul o realitzar tasques que no podíem fer donat el nivell de seguretat activat. Per generar la nostra clau xifrada, necessitarem el mòdul `sha1.ko` de l'API criptogràfica.

Un exemple de la desactivació del nivell de seguretat, seria el següent. Primer generem la clau:

```
:~# echo -n "contrasenya" | openssl sha1
c9ddd8c976785de06c9aa7d72e52a0b6bd05d35e
```

I després, si no teníem el mòdul `sha1.ko`, el carreguem primer i, a continuació, passem la clau generada al mòdul `seclvl.ko`:

```
:~# insmod /lib/modules/2.6.12.1/kernel/crypto/sha1.ko
:~# insmod /lib/modules/2.6.12.1/kernel/crypto/seclvl.ko
                sha1_passwd=c9ddd8c976785de06c9aa7d72e52a0b6bd05d35e
```

Seguint aquest procediment, tenim una manera de poder descarregar el mòdul sense haver de tornar a engegar la màquina. Per fer-ho, només caldrà subministrar-li la contrasenya al mòdul, i ens rebaixarà el nivell de seguretat a 0.

```
:~# echo -n "contrasenya" > /sys/seclvl/passwd
```

Per últim, només recordar que si instal·lem de manera permanent el mòdul al kernel amb alguna utilitat de configuració de mòduls com ara el `modconf(8)` de la distribució Debian GNU/Linux, haurem d'eliminar la línia del mòdul al fitxer `/etc/modules` abans de tornar a engegar la màquina, perquè si no s'activarà per defecte i pot donar conflictes a l'hora de carregar d'altres mòduls del kernel.

5.- Linux Capabilities

En aquest apartat veurem les Linux Capabilities que ja hem vist en algun moment del treball. Al kernel 2.6 de Linux, fruit del treball fet pel projecte LSM, vénen incloses en forma de mòdul del kernel. Tot i així, el concepte de les Capabilities no és nou, i ve essent implementat als kernel Linux des de la seva versió 2.2.

Per tant, conscients de la seva existència en forma de mòdul LSM, farem una explicació del funcionament de les Linux Capabilities al marge de quina sigui la seva implentació. La novetat en els kernels 2.6 és la seva integració en forma de mòdul i els detalls d'aquest procés d'integració han estat breument explicats a la secció 3.1.2. En aquesta secció entrarem a explicar amb més detall el funcionament de les Linux Capabilities i els seus orígens.

• POSIX Capabilities:

Per tal de comprovar els permisos dels processos que s'executen en un sistema provinent del món UNIX, s'estableixen dues categories de processos: els privilegiats i els no privilegiats. Els primers són els llançats per l'usuari `root` (`euid = 0`) i eviten qualsevol control de permisos, mentre que els no privilegiats estan condemnats a passar per una sèrie de comprovacions basades en el sistema de permisos tradicional dels sistemes *NIX, vist als conceptes previs del treball. Aquesta divisió, molt útil per algunes situacions, pot tornar-se perillosa si algú aconsegueix la contrasenya del `root`, i obté els permisos de superusuari. Com hem vist, es tracta d'un model DAC que esdevé molt vulnerable en la situació descrita.

Amb les POSIX Capabilities es va voler proporcionar una divisió en petits conjunts de privilegis de la totalitat dels permisos adquirits quan som al *kernel-land*. D'aquesta manera, serà possible retallar els permisos de `root` en màquines estables, evitant així que un usuari maliciós amb permisos de `root` pugui causar estralls al sistema.

• Origen:

Per tal d'intentar solucionar aquestes limitacions, a mitjan de la dècada dels 80 es començà a desenvolupar l'estàndard POSIX 1003.1e, també conegut com a POSIX 1.e , IEEE 1003.1e i POSIX6. Aquest estàndard volia integrar-se a la interfície del programador (API) POSIX, definida per l'IEEE (*Institute of Electrical and Electronics Engineers*) i

especificada formalment al document IEEE 1003.

Els objectius del POSIX 1003.1e eren proporcionar als sistemes oberts una sèrie d'interfícies de seguretat de llistes de control d'accés (ACL), auditoria, separació de privilegis (*capabilities*), control d'accés obligatori (MAC) i mecanismes d'etiquetes d'informació.

Després de desenvolupar-se durant més de 10 anys, el darrer esborrany de l'estàndard va ser publicat l'octubre de 1997, però no va arribar mai a passar d'esborrany i, finalment, el 1999, POSIX va retirar la proposta d'estàndard 1003.1e, per considerar que no s'adequava a les necessitats ni exigències del món real (havien passat 13 anys des de que s'havia iniciat la definició de l'estàndard), per la falta d'implementacions existents, i per la descoordinació que hi havia entre els grups de treball de l'estàndard[1].

Malgrat tot, existeixen algunes implementacions del POSIX 1003.1e, o d'algunes parts del mateix. Nosaltres estudiarem la implementació a GNU/Linux, però se n'han implementat per FreeBSD i per Solaris

• Linux Capabilities:

Al fitxer de capçalera del kernel `<include/linux/capability.h>`, és on es defineixen les capacitats que poden anul·lar-se i activar-se. N'hi ha un total de 32, de les quals 9 provenen de l'esborrany POSIX 1003.1e i la resta han estat definides específicament per a funcionar als nuclis Linux.

Amb la introducció de les *capabilities* a partir dels kernels 2.2, van dividir-se els privilegis de `root` en aquestes 32 capacitats, i se li assignen unes capacitats determinades (normalment 27), que poden modificar-se un cop el sistema operatiu està en marxa, però no poden ser canviades fins que es torni a engegar l'ordinador. Això ens permet retallar privilegis de `root` en màquines en funcionament 24x7, eliminant possibles vulnerabilitats degudes a un compte `root` compromès.

1 vegeu http://lists.nas.nasa.gov/archives/ext/linux-security-audit/1999/06/msg00057.html per una explicació més extensa

- La divisió de poders:

El llistat de capacitats en què han estat dividits els poders de l'usuari `root` està definit (i mínimament explicat) al fitxer `<include/linux/capability.h>`, i és el següent:

POSIX Capabilities:

`CAP_CHOWN`	deixa realitzar canvis sobre permisos de fitxers via `chown(2)`.
`CAP_DAC_OVERRIDE`	esquiva les comprovacions de permisos de lectura, escriptura i execució sobre els fitxers
`CAP_DAC_READ_SEARCH`	esquiva les comprovacions de lectura sobre els fitxers i d'execució i lectura sobre els directoris
`CAP_FOWNER`	esquiva les comprovacions de permisos en operacions que solen requerir que l'`uid` del procés coincideixi amb l'`id` del propietari del fitxer, excepte quan la capacitat `CAP_FSETID` és aplicable. No se salta cap restricció MAC o DAC.
`CAP_FSETID`	No altera el bits `set-user-ID` i `set-group-ID` quan un fitxer és modificat. Permet la modificació del bit `set-group-ID` d'un fitxer encara que el `gid` d'aquest no coincideixi amb el (s) `gid(s)` proporcionat(s) pel procés actual.
`CAP_FS_MASK`	emprat per triar entre l'ús de l'antiga `suser()` o `fsuser()`. Aquesta opció només és vàlida per a determinades arquitectures, entre les qual no hi ha la x86.
`CAP_KILL`	evita comprovacions quan s'envien senyals amb `kill(2)`.
`CAP_SETGID`	permet la manipulació de `gids` de processos i llistes de `gids` (vegeu `setgid(2)` i `setgroups(2)`). Permet gids falsificats en el pas de credencials de *sockets* del domini UNIX.
`CAP_SETUID`	permet manipulacions d'`uids` de processos (vegeu `setuid(2)`, `setreuid(2)`, `setresuid(2)` i `setfsuid(2)`). Permet `uids` falsificats en el pas de credencials de *sockets* del domini UNIX.

Linux Capabilities específiques:

`CAP_SETPCAP`	permet transferir capacitats del *permitted set* a qualsevol procés i eliminar capacitats del mateix conjunt quan ens ho reclami qualsevol `pid`.
`CAP_LINUX_IMMUTABLE`	permet la modificació dels atributs de fitxer `S_IMMUTABLE` i `S_APPEND` (vegeu `chattr(1)`).
`CAP_NET_BIND_SERVICE`	permet associar ports reservats (els menors que 1024) a un domini d'Internet.
`CAP_NET_BROADCAST`	(no utilitzada) permet *broadcasting* a les *sockets* i escoltes en *multicast*.
`CAP_NET_ADMIN`	permet realitzar determinades operacions de xarxa com ara la modificació de taules d'enrutament, establir opcions privilegiades sobre *sockets*, entrar en mode promiscu, entre d'altres.
`CAP_NET_RAW`	permet l'ús de sockets de tipus `RAW` i `PACKET`.
`CAP_IPC_LOCK`	permet el bloqueig de memòria compartida (`mlock(2)`, `mlockall(2)`, `mmap(2)`, `shmctl(2)`).
`CAP_IPC_OWNER`	esquiva les comprovacions realitzades sobre objectes IPC System V.
`CAP_SYS_MODULE`	permet carregar i descarregar mòduls del kernel. Permet modificacions al *capability bounding set*.
`CAP_SYS_RAWIO`	permet operacions d'E/S (`iopl(2)` i `ioperm(2)`) i l'enviament de missatges USB usant `/proc/bus/usb`.
`CAP_SYS_CHROOT`	autoritza l'ús de `chroot(2)`.
`CAP_SYS_PTRACE`	permet traçar processos amb `ptrace(2)`.
`CAP_SYS_PACCT`	permet crides a `acct(2)`.
`CAP_SYS_ADMIN`	permet la realització d'un ampli ventall d'operacions relacionades amb l'administració del sistema, des del muntatge de dispositius físics fins a la manipulació d'objectes IPC System V (vegeu `<include/linux/capability.h>` per obtenir un llistat complet).
`CAP_SYS_BOOT`	autoritza l'ús de `reboot(2)`.
`CAP_SYS_NICE`	permet la modificació del valor *nice* dels processos (`nice(2)` i `setpriority(2)`). Permet canviar l'algorisme de planificació utilitzat pel procés en curs; n'hi ha tres: temps compartit (per defecte), FIFO i *round-robin* (`sched_setscheduler(2)` i `sched_setparam(2)`). Autoritza l'ús de la funció `sched_setaffinity(2)`.

`CAP_SYS_RESOURCE`	permet les següents operacions: ús d'espais reservats a ext2, crides a `ioctl(2)` per controlar l'ext3 *journaling*, sobreescriptura de les quotes de disc, modificació dels límits dels recursos (`setrlimit(2)`), modificació les restriccions per tamany de les cues de missatges IPC (`msgop(2)` i `msgctl(2)`), modificació del nombre màxim de consoles i mapes de teclat.
`CAP_SYS_TIME`	autoritza les modificacions sobre el rellotge del sistema (sigui lògic o físic) amb les funcions `settimeofday(2)`, `stime(2)` i `adjtimex(2)`.
`CAP_SYS_TTY_CONFIG`	permet crides a `vhangup(2)`.
`CAP_MKNOD`	(a partir del kernel 2.4) permet la creació de fitxers especials via `mknod(2)`.
`CAP_LEASE`	(a partir del kernel 2.4) permet l'establiment de la tinença o propietat d'un fitxer qualsevol (vegeu `fcntl(2)`).
`CAP_AUDIT_WRITE`	(a partir del kernel 2.6.11-rc2) juntament amb `CAP_AUDIT_CONTROL`, serveixen per dur un control d'accés sobre els processos als quals van dirigits els missatges d'auditoria enviats via *netlink*.
`CAP_AUDIT_CONTROL`	(a partir del kernel 2.6.11-rc2) juntament amb `CAP_AUDIT_WRITE`, substitueixen els controls al codi d'auditoria que feia `CAP_SYS_ADMIN` antigament (vegeu el fitxer `<kernel/audit.c>`).

Les capacitats assignades a l'usuari `root` poden consultar-se amb la utilitat `lcap(8)`[1]. Amb la mateixa utilitat podrem deshabilitar capacitats per tal de limitar el poder del superusuari. Cal tenir en compte que, pel propi funcionament de les *capabilities*, un cop deshabilitada una capacitat, no es podrà tornar a habilitar fins que no tornem a engegar l'ordinador. Això és així degut a que les capacitats es van heretant a partir del primer procés que s'executa quan s'inicia el sistema operatiu, l'`init(8)` i es van propagant per la resta de processos llançats a partir d'aquest, seguint un algorisme que veurem més endavant.

- Característiques i tipus de capacitats:

Per a les implementacions actuals del nucli, les Linux Capabilities afecten únicament als processos i als fitxers executables. Malgrat això, teòricament i mitjançant algunes modificacions al programa `login(1)`, podem implementar certs rols d'usuari amb unes

1 llibreria disponible a http://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/ o en forma de paquet, en funció de la distribució GNU/Linux utilitzada.

capacitats desitjades. Nosaltres ens centrarem en les capacitats sobre processos i els fitxers executables, que són les que estan implementades.

- Les capacitats dels processos:

Per a cada procés es defineixen tres conjunts de mapes de bits (*bitmaps sets*), que són les capacitats heretades (*inherited*, I), permeses (*permitted*, P) i efectives (*effective*, E). Cadascuna d'aquestes capacitats és implementada com un bit a cadascun dels *bitmaps*, que pot estar a activat o no. Les capacitats es distingueixen de la següent manera:

- Efectives: capacitats utilitzades pel kernel a l'hora de comprovar els permisos del procés.

- Permeses: capacitats que el procés pot prendre o utilitzar. Determinen què pot fer realment el procés. Si un procés s'elimina una capacitat d'aquest conjunt, no podrà tornar-la a utilitzar més (excepte quan executa un programa propietat del `root` i amb el bit `setuid` activat).

- Heretades: capacitats obtingudes pel procés actual després de fer una crida a `execve(2)`. Els processos creats amb un `fork(2)` hereten una còpia dels conjunts de capacitats del seu pare.

Quan un procés vol realitzar una operació privilegiada, el kernel s'encarrega de comprovar el bit adequat al conjunt de capacitats efectives del procés, en comptes de verificar que l'`euid` d'usuari sigui 0, com s'havia fet abans de la introducció de les *capabilities*. Per exemple, si un procés vol afegir un mòdul al kernel, el propi kernel comprovarà que el procés tingui el bit `CAP_SYS_MODULE` activat (actualment és el bit 16) al conjunt de capacitats efectives.

Un procés pot tenir capacitats activades al *permitted set* que no ho estan a l'efectiu. Això indica que les capacitats absents a l'*effective set* han estat deshabilitades temporalment. Aquesta distinció entre capacitats permeses i efectives es fa per tal de permetre als processos aïllar operacions que necessiten privilegis.

En les implementacions actuals del nucli, un procés té concedides les mateixes capacitats efectives i permeses quan és resultat de l'execució d'un programa propietat del `root` i amb el bit `setuid` activat, o bé quan un procés amb un `euid = 0` executa un nou programa.

- Pas de capacitats entre processos:

Hem vist a través de l'*inherited bitmap* que els processos hereten les capacitats de qui els ha llançat mitjançant una crida a `execve(2)`, i que un procés creat a partir d'un `fork(2)` rep una còpia exacta de les capacitats del seu pare. Aquest és el procediment que utilitza el kernel per transferir capacitats entre processos, i ho fa per via de la crida al sistema `capset(2)`, que rep dues estructures de dades que contenen informació sobre el procés i les *capabilities* que li volem assignar. Així, tot procés té un *bitmap set* de capacitats i el kernel podrà fer les comprovacions oportunes en que el procés sol·liciti operacions privilegiades.

Un procés podrà manipular les seves pròpies capacitats gràcies a `capset(2)` i, en cas de tenir activada la capacitat `CAP_SETPCAP`, també podrà modificar el conjunt de capacitats de qualsevol altre procés. De totes maneres, en cas de voler desenvolupar aplicacions que utilitzin les Linux Capabilities, es recomana l'ús de funcions més amigables, com són les proporcionades amb les fonts de la llibreria `libcap` (vegeu `capsetp(3)`).

En els sistemes convencionals, com a opció per defecte, la capacitat `CAP_SETPCAP` està deshabilitada. Per tal d'eliminar aquesta restricció haurem de modificar les definicions de les constants `CAP_INIT_EFF_SET` i `CAP_INIT_INH_SET` que podrem trobar al fitxer de capçalera `include/linux/capability.h` al directori que conté les fonts del kernel, de manera que on hi posa:

```
#define CAP_INIT_EFF_SET    to_cap_t(~0 & ~CAP_TO_MASK(CAP_SETPCAP))
```

ho canviarem per:

```
#define CAP_INIT_EFF_SET    to_cap_t(~0)
```

També haurem de canviar la línia on hi posa:

```
#define CAP_INIT_INH_SET    to_cap_t(0)
```

per aquesta altra:

```
#define CAP_INIT_INH_SET    to_cap_t(~0)
```

Un cop fetes aquestes modificacions, caldrà recompilar el nucli per a que aquest prengui els nous valors durant l'arrencada del sistema.

- El *capability bounding set*:

Aquest concepte serveix per implementar una forma ràpida de comprovar els privilegis d'un procés. El *capability bounding set* és un valor (en format decimal amb signe) guardat al fitxer `/proc/sys/kernel/cap-bound`. Quan un programa és executat, es realitza un AND lògic entre aquest valor i les capacitat permeses i efectives del procés, de manera que podem establir una manera de controlar les capacitats dels processos executats posteriorment.

L'únic procés que hauria de poder afegir bits al *capability bounding set* és l'`init(8)`, mentre que l'usuari `root` només hauria de poder esborrar bits en aquest conjunt (deixar-los a zero per deshabilitar alguna capacitat). D'aquesta manera, un cop deshabilitada una capacitat, fins que no es torni a iniciar el sistema, no es podrà tornar a activar, ja que és l'`init(8)` qui torna a definir el *capability bounding set* cada cop que s'executa.

- Les capacitats dels fitxers (executables):

Per a que el sistema de capacitats sigui totalment efectiu, fa falta que el sistema de fitxers proveeixi d'algun mecanisme per passar les capacitats definides per a un fitxer executable al procés que el crida. El que seria ideal és que els tipus i els mecanismes de transferència de capacitats fossin els mateixos que en el cas dels processos, però malauradament, a les versions actuals del kernel, això no és possible.

El que si que es defineix, és la possibilitat d'associar tres conjunts de capacitats a un fitxer executable que, juntament amb les capacitats dels procés que el llança, permetran determinar quines seran les capacitats del procés després de l'execució del programa.

Aquests tres conjunts de capacitats són:

- Permeses: a les capacitats d'aquest conjunts se'ls hi fa un AND lògic amb l'*inherited set* del procés per tal de determinar les capacitats heretades que se li permetran al procés després de realitzar l'execució.

- Forçades: són les capacitats autoritzades de forma automàtica al procés, sense fer cas de les capacitats heretades.

- Efectives: les capacitats del nou *permitted set* del procés s'han d'establir de la mateixa manera al nou *permitted set* (normalment les capacitats efectives del fitxer són o tot zeros o tot uns).

El procediment que es segueix quan es llança l'execució d'un programa és el següent:

- Els tres conjunts de capacitats dels fitxers s'inicialitzen a zero.
- Si es tracta de l'execució d'un programa amb el bit `set-uid` activat o l'`eid` del procés és el de `root`, els conjunts de capacitats permeses i forçades del fitxers són posats tots a uns.
- Si es tracta de l'execució d'un programa amb el bit `set-uid` activat, el conjunt de capacitats efectives també és posat tot a uns.

Llavors, el kernel calcula les noves capacitats del procés utilitzant els següent algoritme:

```
P'(permitted) = (P(inherited) & F(allowed)) | (F(forced) & cap_bset)

P'(effective) = P'(permitted) & F(effective)

P'(inherited) = P(inherited)
```

on:

`P` indica el valor del conjunt de capacitats del procés abans de l'execució.

`P'` indica el valor del conjunt de capacitats del procés després de l'execució.

`F` indica el conjunt de capacitats del fitxer.

`cap_bset` és el valor del *capability bounding set*.

- Resumint: com treballem amb les capacitats?

Hem vist que les capacitats afecten als processos i als fitxers executables, i que segons estiguem parlant d'un o de l'altre, el kernel utilitza mètodes diferents per a treballar amb les capacitats. En el cas dels processos les comprovacions es fan a través del *capability bounding set*, mentre que quan parlem de fitxers executables s'utilitzarà l'algoritme que acabem de veure.

Però com a usuaris, el que ens interessa és deshabilitar determinats privilegis de `root`. Per fer-ho de manera ràpida i senzilla, disposem de la utilitat `lcap` que ens ho deixa fer. Si no, també tenim la possibilitat de modificar directament el fitxer que conté el *capability bounding set*. Això últim ho podrem fer, com a usuari `root`, executant des de la línia de

comandes:

```
:~# echo $((~0 & ~(1 << VALOR))) > /proc/sys/kernel/cap-bound
```

on VALOR és el valor decimal que pren la capacitat que volem deshabilitar segons es defineix a `<include/linux/capability.h>`.

Cal recordar que tant si volem desenvolupar aplicacions que utilitzin les Linux Capabilities com per disposar de totes les capacitats activades, caldrà modificar els valors de les constants `CAP_INIT_EFF_SET` i `CAP_INIT_INH_SET` com hem vist a la secció de pas de capacitats entre processos.

- Instal·lació:

Per activar el mòdul de les Linux Capabilities anirem, al menú de configuració del kernel, a la secció *Security Options -> Default Linux Capabilities* on podrem triar entre compilar el mòdul a dins del kernel o fer-ho com a mòdul carregable. Si escollim aquesta segona opció, se'ns compilen dos mòduls, el `commoncap.ko` i el `capability.ko` i caldrà carregar-los en aquest ordre, mitjançant l'utilitat `insmod(8)` de la següent manera:

```
:~# insmod /lib/modules/2.6.12.1/kernel/security/commoncap.ko
:~# insmod /lib/modules/2.6.12.1/kernel/security/capability.ko
```

- Un cas pràctic:

És conegut que una de les vulnerabilitats més explotades als sistemes GNU/Linux és la inserció de mòduls del kernel per tal d'aconseguir accedir a l'espai de nucli i poder operar-hi sense cap mena de restricció (els coneguts LKM). Això pot succeir en màquines on el compte de `root` ha estat compromès i algun usuari pot accedir-hi amb les capacitats assignades per defecte (que inclouen la capacitat per inserir mòduls del kernel). Executant, com a root, la instrucció:

```
:~# lcap 16
```

Haurem eliminat la possibilitat que el superusuari insereixi mòduls al kernel, i evitarem la vulnerabilitat. L'inconvenient d'això és que quan l'administrador del sistema vulgui inserir algun mòdul nou al kernel, haurà de reengegar el sistema, inserir el mòdul i tornar a *deshabilitar* la `CAP_SYS_MODULE`.

Combinant la desactivació de `CAP_SYS_MODULE`, amb la de `CAP_SYS_BOOT` i `CAP_SYS_RAWIO`, el nostre sistema serà molt difícil de modificar per qualsevol usuari, ja que, a part de no poder carregar mòduls maliciosos, no serà possible accedir a l'espai de memòria del kernel ni podrem tornar a engegar la màquina.

6.- NSA SELinux

SELinux prové de Security-Enhanced Linux, i és una modificació del kernel Linux feta per un departament de la *National Security Agency* (NSA) dels Estats Units anomenat *Information Assurance*[1]. Un grup d'investigadors d'aquest departament va publicar, l'any 2000, la primera versió SELinux i des de llavors ha anat publicant actualitzacions per a les diferents versions del kernel.

A la secció 4 d'aquest treball hem parlat dels Linux Security Modules, i hem vist que va ser precisament l'aparició de SELinux el que va donar una empenta al naixement del projecte LSM. Des de mitjan de 2001 que existeix un prototipus SELinux que utilitza els Linux Security Modules i la integració de SELinux al kernel 2.6 en forma de mòdul de seguretat és completament estable .

En aquest apartat farem una introducció al projecte SELinux, la seva implementació en forma de mòdul LSM així com quina és la política de seguretat per defecte que s'activa en instal·lar aquest mòdul al kernel.

• Introducció:

Tal i com es comenta a la documentació de SELinux, aquest és un prototipus d'investigació del kernel Linux que, juntament amb un conjunt d'utilitats amb funcionalitats de seguretat reforçades o millorades, volen demostrar a la comunitat d'usuaris de GNU/Linux la validesa del control d'accés obligatori per tothom (MAC) i com poden implementar-se aquest tipus de control d'accés.

SELinux imposa polítiques MAC que cedeixen a les aplicacions de l'espai d'usuari (siguin programes d'usuari o aplicacions de servidor) la quantitat mínima de privilegis necessaris per què puguin funcionar. Limitant els programes d'aquesta manera es redueix o s'elimina la capacitat de causar danys al sistema, siguin producte d'un error de disseny o mala configuració del programa, o fruit d'un acte malintencionat d'algun usuari. Aquests mecanismes de limitació de permisos actuen de forma independent al control d'accés tradicional (DAC) existent a GNU/Linux. SELinux no té concepte de `root` ni depèn de les mancances dels mecanismes de seguretat coneguts a GNU/Linux com poden ser la dependència de les comprovacions dels bits `setuid` i `setgid`.

1 http://www.nsa.gov/ia/index.cfm

Les noves característiques oferides per SELinux van ser dissenyades per fer complir la separació de la informació en base a requeriments de privadesa o confidencialitat i d'integritat de les dades. Volen impedir que els processos llegeixin o facin malbé dades i programes, se saltin els mecanismes de seguretat per aplicacions, executin programes poc dignes de confiança o interfereixin amb d'altres programes, violant les polítiques de seguretat del sistema. SELinux també pot ser útil a l'hora de limitar el dany potencial que poden provocar programes maliciosos o defectuosos, així com per habilitar un sistema d'autoritzacions de seguretat que permet accedir a diferents tipus d'informació que requereixen certs privilegis, sense que això comprometi la seguretat del sistema. Aquesta darrera característica s'ha d'entendre tenint en compte el context de la NSA i la quantitat d'informació conficendial i governamental que gestiona.

El control d'accés a SELinux es realitza amb la combinació de dos tipus de control d'accés: Type Enforcement[1] (TE) i Role-Based Access Control[2] (RBAC). Els components TE d'aquesta combinació defineixen un conjunt ampliable de dominis i de tipus. Cada procés té un domini associat i cada objecte del kernel té un tipus associat. El domini indica la funció que té la intenció de realitzar el procés i el tipus indica la finalitat de l'objecte. El fitxers de configuració de SELinux especifiquen, mitjançant taules indexades per domini i tipus, els requeriments necessaris per a que a un procés d'un domini pugui realitzar la funció que tenia la intenció de fer. També especifiquen com poden accedir els dominis als tipus i com poden interactuar amb d'altres dominis, així com les transicions permesen entre dominis. Amb aquests components podem definir canvis de dominis automàtics quan programes d'un tipus són executats, garantint que els processos del sistema i determinats programes retornen als seus respectius dominis de forma automàtica quan són executats.

Pel que fa als components RBAC, defineixen un conjunt ampliable de rols d'usuari de manera que cada procés té un rol associat. D'aquesta forma garantim que els processos de sistema i aquells usats per la seva administració puguin ser separats dels que són executats per usuaris planers. Als fitxers de configuració podrem establir els dominis als quals poden accedir cadascun dels rols. Per defecte existeixen dos rols definit per als usuaris, el `user_r` per a usuaris normals i el `sysadm_r` per als administradors del sistema, que són activats a través del programa `login(1)`.

A continuació veurem com s'implementen al kernel aquestes millores en la seguretat de GNU/Linux i quines són les polítiques per defecte de les quals disposem en activar el mòdul NSA SELinux al nucli.

1 http://www.securecomputing.com/index.cfm?skey=738
2 http://csrc.nist.gov/rbac/

• Implementació de SELinux:

Originàriament desenvolupada en forma de *kernel patch*, la implementació de SELinux va ser adaptada[1] per poder ser integrada a l'entorn LSM. Nosaltres ens centrarem en la implementació dins d'aquest framework, explicant els conceptes bàsics de la mateix, la seva arquitectura interna, així com l'API i els tipus de funcions que proporciona. També veurem com s'implementen els ganxos LSM disponibles per als diferents objectes del kernel.

El codi font de SELinux el podem trobar a `<security/selinux/>` dins del directori que conté les fonts del kernel. Prendrem com a base aquesta ruta quan anomenem fitxers que implementen SELinux, a no ser que indiquem el contrari.

• Conceptes bàsics de SELinux:

SELinux es basa en l'arquitectura de seguretat del sistema operatiu Flask[2] (*Flux Advanced Security Kernel*), dissenyat per investigadors de la pròpia NSA i de l'empresa Secure Computer Corporation[3] i que proporciona un suport flexible per definir polítiques de seguretat i mecanismes de control d'accés tipus MAC. Com a herència de Flask, SELinux separa de forma clara les polítiques d'execució de codi de les polítiques de presa de decisions.

La **política de presa de decisions** és encapsulada en un component separat del sistema operatiu anomenat servidor de seguretat (*security server*). Aquest, s'ajuda d'un vector anomenat Access Vector Cache (AVC), que disposa d'una memòria cau del càlcul de les decisions d'accés preses pel servidor de seguretat i que intenta minimitzar la sobrecàrrega del sistema fruit de les consultes constants a les polítiques de seguretat.

Pel que fa a la **política d'execució de codi**, s'integra dins del mateix codi del kernel que controla objectes com ara els sistemes de fitxers, les *sockets* o el System V IPC. Les decisions de seguretat quan s'està executant codi vénen donades des del servidor de seguretat i l'AVC, i serveixen per afegir etiquetes de seguretat als processos i objectes del kernel i per controlar les operacions basades les etiquetes o *security labels*.

Existeixen dos tipus de dades indepentents a les polítiques de seguretat que ens permeten definir aquestes etiquetes de seguretat: el context de seguretat i l'identificador de seguretat o SID. El context de seguretat és una cadena de text que representa

1 podeu trobar una explicació dels canvis realitzats sobre la implementació original de SELinux per adaptar-lo al *framework* LSM aquí: http://www.nsa.gov/selinux/papers/module/x91.html
2 http://www.cs.utah.edu/flux/fluke/html/flask.html
3 http://www.securecomputing.com/

l'etiqueta de seguretat, mentre que el SID és un manegador (*handler*) intern associat pel *security server* a un context de seguretat. Aquests dos tipus de dades són gestionats directament per la política d'execució de codi, que és qui associa els SID als processos i objectes actius, consulant al servidor de seguretat sempre que s'hagin de calcular nous SID. A més, es fa una distinció entre els SID del kernel i els de l'espai d'usuari, de manera que els SID del kernel no s'exporten mai al *user-land* i els mecanisme que fan complir les polítiques definides disposen de llistats propis de SID gràcies a un AVC de l'espai d'usuari que s'inclou a la llibreria `libselinux`.

Per realitzar comprovacions de permisos sobre les operacions, la política d'execució de codi consulta l'AVC (i aquest, si no té una resposta, consulta al *security server*), subministrant-li dos SID (un d'origen i l'altre de destí) juntament amb una classe de seguretat o *security class* que identifica el tipus d'objecte implicat en l'operació. Aquesta *security class* té associada un conjunt de permisos que són els utilitzats per realitzar les comprovacions de control d'accés.

- Arquitectura Interna:

Explicarem a continuació quin és l'esquelet SELinux. Aquest mòdul de seguretat està format per sis elements principals que són: el servidor de seguretat, el vector AVC, la taula d'interfície de xarxa, el codi de notificació d'events *netlink*, el pseudosistema de fitxers `selinuxfs` i les implementacions dels ganxos LSM a funcions de seguretat.

Hem vist per sobre el ***security server*** i ja sabem que proporciona una interfície que centralitza la presa de decisions i permet a la resta d'elements del mòdul actuar de forma independent a les polítiques de seguretat definides. La definició d'aquestes interfícies la podem trobar al fitxer de capçalera `include/security.h`, al directori del mòdul. Al mòdul SELinux distribuït amb el kernel 2.6 s'hi inclou un servidor de seguretat d'exemple que implementa la combinació TE/RBAC descrita a l'introducció d'aquesta secció i que veurem amb una mica més de detall quan parlem de les polítiques de configuració.

El **vector AVC** proporciona una memòria cau del càlcul de decisions d'accés obtingudes del servidor de seguretat que busca minimitzar la sobrecàrrega del sistema. També proporciona interfícies als ganxos LSM per tal de fer les comprovacions de permisos de la forma més eficient possible, així com interfícies pel servidor de seguretat que li permetin gestionar la memòria cau. El codi font de l'AVC pot trobar-se al fitxer `avc.c`, mentre que les interfícies pels ganxos són a `include/avc.h` i les del servidor de seguretat a `include/avc_ss.h`.

La **taula d'interfície de xarxa** conté la correspondència entre els dispositius de xarxa i els contexts de seguretat. Aquesta taula al marge del servidor de seguretat (recordem que era ell qui duia la gestió de les relacions entre SID i contexts de seguretat) va ser

implementada per poder realitzar la integració a LSM i permet als ganxos LSM consultar i obtenir els SID associats a un dispositiu de xarxa qualsevol. Les funcions que permeten aquest tipus de consultes i operacions pot trobar-se a `include/netif.h`, mentre que el codi de la pròpia taula d'interfície de xarxa és al fitxer `netif.c`.

El **codi de notificació d'events *netlink*** permet que el mòdul SELinux notifiqui als processos actualitzacions en les polítiques de seguretat. Les notificacions són emprades per l'AVC de l'espai d'usuari per mantenir estats consistents amb el kernel. El codi font d'aquest component de SELinux pot trobar-se al fitxer `netlink.c`.

Pel que fa al **pseudosistema de fitxers `selinuxfs`**, és l'encarregat d'exportar l'API de polítiques del servidor de seguretat als processos. Fruit de les modificacions que es van haver de fer a l'API original de SELinux per adaptar-la a la branca 2.6 del kernel, va ser descomposada en tres components encapsulats a la llibreria `libselinux`: atributs de processos, atributs de fitxers i API de polítiques. El pseudosistema de fitxers `selinuxfs` és qui dóna suport a les crides de l'API de polítiques. Podrem trobar el codi font de `selinuxfs` al fitxer `selinuxfs.c`.

Per últim, els **ganxos a funcions** gestionen la informació de seguretat associada als objectes del kernel i fan un control d'accés per cada operació d'aquest. Per mitjà de crides al servidor de seguretat i a l'AVC, els ganxos són coneixedors de les decisions de seguretat i actuen en funció d'aquestes, etiquetant i controlant els objectes del kernel. En el cas dels fitxers, també es fan crides als atributs extesos del sistema de fitxers per tal d'obtenir i establir contextos de seguretat sobre aquest tipus d'objecte. El codi font dels ganxos podem trobar-lo al fitxer `hooks.c` i les estructures de dades per emmagatzemar-hi la informació de seguretat associada als objectes del kernel estan definides a ls fitxer de capçalera `include/objsec.h`.

- Inicialització del mòdul SELinux:

Per poder carregar el mòdul de seguretat SELinux disposem d'un seguit de funcions que ho fan en diferents passos. Com que de vegades es fa necessari que el kernel estigui totalment inicialitzat per tal d'aplicar determinades polítiques i per evitar conflictes amb d'altres mòduls (siguin de seguretat o no), el procés de càrrega està dividit en sis funcions: `selinux_init`, `selinux_nf_ip_init`, `sel_netif_init`, `selnl_init`, `init_sel_fs` i `selinux_complete_init`.

La càrrega del mòdul SELinux comença amb `selinux_init`, que és l'encarregada d'establir l'estat de seguretat pel procés `init(8)`, així com d'inicialitzar el vector AVC abans que es realitzi qualsevol comprovació de permisos. Un cop fet això, `selinux_init` estableix el mòdul secundari (ara per ara només hi ha suport pel mòdul *dummy* o el *capabilities*) i, finalment, es registra com a mòdul LSM primari mitjançant una crida a

`register_security` que hem vist abans, quan parlàvem del registre de mòduls LSM, a la secció 4.1.

Per la inicialització de la seguretat a la xarxa existeixen les funcions `selinux_nf_ip_init`, `sel_netif_init` i `selnl_init` que, respectivament, inicialitza els ganxos a funcions per a NetFilter[1] que fa comprovacions de permisos per als paquets de xarxa sortints, inicialitza la taula d'interfície de xarxa que conté la relació SID-dispositius de xarxa, i inicialitza les *netlink sockets* utilitzades en l'enviament de notificacions a l'espai d'usuari.

Respecte les dues darreres funcions, `init_sel_fs` inicialitza i munta per al kernel el pseudosistema de fitxers `selinuxfs`, mentre que `selinux_complete_init` és l'encarregada de finalitzar tot el procés d'inicialització un cop el procés `init(8)` ha estat carregat.

El fitxer `hooks.c` conté la implementació de les funcions `selinux_init`, `selinux_complete` i `selinux_nf_ip_init`, a `netlink.c` hi trobarem la de `selnl_init`, a `netif.h` la de `sel_netif_init` i, per últim, al fitxer `selinuxfs.c` hi ha el codi de la funció `init_sel_fs`.

- Funcions d'ajuda per als *hooks*:

Per a facilitar la feina als *hooks* definits per SELinux, es proporcionen un seguit de funcions d'ajuda que poden usar-se (i s'usen) en la implementació dels ganxos. A banda de les usades per cadascun dels ganxos, implementades a mode de mètodes privats, poden definir-se tres grups de funcions d'ajuda: funcions per la reserva d'espai de memòria, funcions d'inicialització i funcions de comprovació de permisos.

Per la **reserva de memòria** de les estructures de seguretat definides a `include/objsec.h`, s'utilitzen funcions del tipus `xxxx_alloc_security` i `xxxx_free_security`, implementades al fitxer `hooks.c`, cada cop que es criden les funcions LSM `alloc_security` i `free_security`, respectivament.

Les funcions d'**inicialització** fan referència a la inicialització d'algunes estrutures de seguretat, com poden ser els inodes o bé els superblocs.

Pel que fa al **control de permisos**, existeix un conjunt de funcions d'ajuda que simplifiquen el codi de molts dels ganxos que duen a terme aquesta mena de comprovacions, o que realitzen les invocacions a l'AVC i retornen les respostes als *hooks*. Exemples d'aquest tipus de funcions són `task_has_perm` o `inode_has_perm`, també definides a `hooks.c`.

1 http://netfilter.org/

- Implementació dels hooks:

Com a mòdul LSM que és, SELinux implementa ganxos a funcions pel tractament de diferents objectes del kernel, a més d'incloure els camps de seguretat LSM dins de les estructures de dades que defineixen aquests objectes. Els camps de seguretat LSM es corresponen, a SELinux, amb els SID de cada objecte.

Ja hem vist les funcions d'ajuda que incorpora SELinux per facilitar la feina als *hooks* i que són utilitzades pel tractament de la seguretat de pràcticament tots els objectes del kernel que controla SELinux. A continuació, veurem quins són aquests objectes i, de manera general, quins ganxos s'implementen per a cadascun d'ells i les funcions d'ajuda que utilitzen a part de les que hem vist a l'apartat anterior. Informació més detallada i llistats complets de tots els ganxos definits pot trobar-se a la documentació de SELinux[1] i al codi font de SELinux, buscant per "`selinux_`", que és com comencen tots els noms dels ganxos.

Les estructures de seguretat que controlen els objectes del kernel estan definides a `include/objsec.h` i són de la forma `xxxx_security_struct`. Al fitxer `hooks.c` és on hi trobarem la major part del codi relacionat amb els ganxos.

- Ganxos pels processos:

Per al control de l'execució dels processos, SELinux defineix una estructura anomenada `task_security_struct` on hi defineix cinc camps de seguretat en forma de SID: `osid`, `sid`, `exec_sid`, `create_sid` i `ptrace_sid`.

Els ganxos a funcions per la gestió de la seguretat en els processos permeten, entre d'altres operacions: la inicialització, creació, pausa i destrucció de tasques, les relacionades amb el planificador i nivells de prioritat del procés, obtenir i desar informació dels atributs del procés i el control de relació amb els inodes associats a cada procés via `/proc/pid`.

1 http://www.nsa.gov/selinux/info/docs.cfm#tech

- Ganxos per la càrrega de programes:

En el moment que un procés sol·licita l'execució d'un nou programa al kernel (normalment, via `execve(2)`) també es duen a terme operacions de seguretat. L'estructura que encapsula aquest moment s'anomena `bprm_security_struct` i el *security field* que incorpora es diu `sid`.

Pel que fa als ganxos, n'hi ha per establir el camp de seguretat del procés que crida `execve`, per definir els atributs de seguretat del nou procés i reafirmar els de l'existent i per a realitzat execucions en un entorn segur.

- Ganxos per superblocs:

Les implementacions dels ganxos a funcions per al tractament de superblocs gestionen els camps de seguretat de les estructures del tipus `super_block`, definit al fitxer `<include/linux/fs.h>` i realitzen un control d'accés sobre les operacions amb el sistema de fitxers.

Les funcions proporcionades per SELinux serverixen per controlar les operacions de muntatge i desmuntatge de dispositius de bloc, gestió de quotes de disc i operacions amb dades pertanyents a sistemes de fitxers muntats, així com control de permisos a l'hora d'obtenir atributs del sistema de fitxers.

- Ganxos per inodes:

Aquests ganxos són els emprats per la gestió dels camps de seguretat a l'estructura `inode` i per dur a terme un control d'accés sobre aquest tipus d'objectes. Cal destacar que aquests ganxos han de gestionar inodes situats en els diversos tipus de sistemes i pseudosistemes de fitxers

Els camps de seguretat es defineixen a l'estructura `inode_security_struct`, anomenats `task_sid` i `sid`. També es defineixen un seguit de classes de seguretat (*security classes*) per descriure els diferents tipus d'objectes que pot contenir un inode (fitxer, directori, enllaç, *pipe*, *socket*, fitxer orientat a bloc, fitxer orientat caràcters).

Les funcions d'ajuda ofereixen suport per a la comprovació de permisos, el moment posterior i previ a la creació d'un inode. Pel que fa als ganxos a funcions, n'hi ha per la creació i esborrat de qualsevol classe d'inode, per la obtenció i establiment d'informació de seguretat, per a definir atributs extesos sobre un inode i per fer un control d'accés.

- Ganxos per fitxers:

Com tots els altres ganxos, els existents per fitxers permeten gestionar els camps de seguretat, en aquest cas els definits a l'estructura `file`, a més de realitzar un control d'accés sobre aquest tipus d'objectes.

La informació de seguretat està emmagatzemada a l'estructura `file_security_struct`, que conté dos *security fields*: `sid` i `fown_sid`.

Existeixen ganxos a funcions per al control de la majoria d'operacions que es poden realitzar amb un fitxer obert, com ara validació de permisos, operacions E/S (`ioctl(2)`), realització de *mappings* de memòria, manipulació de descriptors de fitxers (`fcntl(2)`), bloqueig de fitxers, entre d'altres.

- Ganxos pel System V IPC:

Aquest tipus de ganxos permeten la gestió de la seguretat sobre semàfors, segments de memòria compartida i cues de missatges del System V IPC.

A l'estructura `ipc_security_struct` podem trobar-hi la informació de seguretat per aquest tipus d'objectes, d'entre la qual destaca el `sid` i la classe de seguretat associada a l'objecte (`sclass`). A més, es defineix una altra estructura per la informació de seguretat del missatges IPC, anomenada `msg_security_struct`.

Els ganxos a funcions ens permetran controlar tant les operacions relacionades amb l'IPC en general (comprovació de permisos i obtenció d'identificadors d'objectes IPC), com les definides per semàfors, memòria compartida i cues de missatges, i que comprenen accions com ara la creació, destrucció, lectura i escriptura (enviament i recepció en el cas dels missatges) o bloqueig d'aquests objectes.

- Ganxos per sockets:

Les *sockets* de l'espai d'usuari definides a l'`struct socket` tenen associada una estructura inode, que s'aprofita per gestionar la seguretat d'aquest tipus de *sockets*. Pel que fa a les *sockets* de la capa de xarxa (`struct sock`), SELinux defineix un camp de seguretat, tot i que només en recomana l'ús per a *sockets* locals i del domini UNIX, ja que el protocol TCP no garanteix la gestió correcta d'aquest camp de seguretat. També es defineixen classes de seguretat per als diferents tipus de *sockets* (orientades a connexió

o a datagrama, TCP, UDP, *Netlink*, *packet sockets*, *raw sockets* i *key sockets*).

Els ganxos a funcions permeten gestionar les operacions de xarxa més freqüents, com són la creació de *sockets*, connexió, *binding*, escolta i acceptació de connexions, enviament i recepció de dades, desconnexió i d'altres relacionades amb l'obtenció d'informació de la *socket*.

- Ganxos per *IP networking*:

Malgrat l'existència, en forma de *patch*, d'un seguit de camps de seguretat i de ganxos per permetre la gestió de la seguretat en les estructures de dades relacionades amb la xarxa, els camps de seguretat i ganxos LSM per aquest tipus d'objectes no van ser inclosos a la versió 2.6 del kernel. SELinux havia desenvolupat un conjunt de ganxos basant-se en el model LSM, i va haver de redissenyar i reduir els ganxos per l'*IP networking*.

Així, existeixen ganxos per a IPv4 i IPv6, basats en els ganxos a NetFilter, que realitzen controls de permisos per als paquets sortints, després de ser encaminats cap a la xarxa. Els paquets entrants es controlen mitjançant un ganxo per *sockets* com els que hem vist abans (concretament, el `selinux_socket_sock_rcv_skb`).

- Altres ganxos:

Dins d'aquest grup s'hi inclouen la resta de ganxos desenvolupats que no s'escauen a les seccions anteriors, d'entre els quals destaquen: els ganxos relacionats amb les *capabilities* en cas que les habilitem com a mòdul LSM secundari, els ganxos relacionats amb el control sobre les operacions que permeten realitzar canvis de paràmetres del kernel quan aquest s'està executant (vegeu `sysctl(8)`) i, finalment, els ganxos relacionats amb els mecanismes de *logs* del sistema.

- **Política de configuració:**

Amb el mòdul SELinux s'activa una política de configuració de seguretat per defecte. Aquesta configuració engloba des de la definició de dominis i tipus TE i la definició de rols d'usuari basat en RBAC, fins a la definició de contextos de seguretat. La política de configuració la podrem trobar, un cop haguem instal·lat SELinux i les utilitats i llibreries que requereix, a partir del directori `/etc/selinux/`.

Hem vist a la introducció d'aquesta secció que per cada procés hi ha un domini associat i cada objecte té un tipus associat, i que és la política de seguretat qui especifica els accessos permesos a cadascun dels objectes per part dels dominis i quines relacions entre dominis es poden establir. Els tipus es poden agrupar en set grups principals, cadascun dels quals està definit en un fitxer amb extensió `*.te`. Aquests tipus permeten definir objectes relacionats amb: seguretat (`security.te`), dispositius (`device.te`), fitxers (`file.te`), pseudosistema de fitxers `/proc` (`procfs.te`), fitxers a `/dev/pts` (`devpts.te`), NFS (`nfs.te`) i xarxa (`network.te`). Pel que fa als dominis, n'hi ha quatre de definits, que són els relacionats amb: qualsevol procés (`every.te`), processos de sistema (`system.te`), programes d'usuari (es defineixen uns quants fitxers, per diferents aplicacions d'usuari) i *login* d'usuaris (`user.te` i `sysadm.te`).

També caldrà definir els rols d'usuari permesos al sistema i rols d'objectes. Per defecte, se'n defineixen quatre, que són: `object_r` per objectes, `system_r` per processos de sistema, `user_r` per processos d'usuari no privilegiats i `sysadm_r` per l'administrador del sistema. La política de seguretat defineix quins dominis poden ser accedits per cadascun dels rols, i als rols se'ls hi assigna un domini per defecte. A mesura que l'usuari va executant programes, els canvis de domini es faran automàticament.

Amb aquest nivell de granuralitat tant elevat és possible aconseguir una sèrie d'objectius de seguretat, llistats a continuació:

- Controlar l'accés en cru (*raw access*) a les dades del sistema, mitjançant la definició de tipus per dispositius de memòria del kernel, dispositius de disc i `/proc/kcore`, i definint dominis separats (amb control d'accés) per accedir a aquests tipus.

- Protegir la integritat del kernel. Es defineixen tipus per a fitxers d'arrencada, fitxers objecte de mòduls, utilitats de mòduls, fitxers de configuració de mòduls i paràmetres `sysctl(8)`, al mateix temps que es defineixen els dominis necessaris per accedir a aquesta mena de tipus.

- Protegir la integritat del programari de sistema, la informació de configuració del sistema i els *logs* del sistema. Hi ha tipus definits per llibreries i binaris de sistema per poder controlar l'accés a aquests fitxers.

- Limitar el dany potencial que pot ser causat per l'explotació d'una vulnerabilitat en un procés que requereix privilegis, sigui un procés de sistema o un programa amb el bit `setuid` o `setgid` activat. Aquest tipus de processos són col·locats en dominis separats, concedint-los-hi únicament els permisos que necessiten per funcionar.

- Evitar que els processos privilegiats executin codi maliciós. La política de configuració defineix un tipus executable per cada procés privilegiat i només permet transicions al domini privilegiat executant aquest tipus. El domini administrador pot executar programes creats per d'altres administradors així com programari de sistema, però no pot fer-ho amb programes creats per usuaris planers o per processos de sistema.

- Evitar entrades al rol i al domini administrador sense autenticació prèvia. L'accés a aquests domini i rol només és possible a través del programa `login(1)`.

- Evitar la interferència dels processos d'usuari amb els de sistema o els de l'administrador, controlant l'ús del pseudosistema sistema de fitxers `procfs`, de la funció `ptrace(2)` i de les senyals.

- Per últim, protegir als usuaris i als administradors de l'explotació de debilitats del navegador Netscape per part de codi maliciós present a les pàgines web visitades. La política de configuració aïlla aquest navegador en un domini separat.

• Instal·lació:

A banda del mòdul que es proporciona amb el kernel, el codi de SELinux consta d'una llibreria per la manipulació binària de les polítiques (`libsepol`), un compilador de polítiques (`checkpolicy`), una altra llibreria per aplicacions de seguretat (`libselinux`), utilitats relacionades amb les polítiques (`policycoreutils`) i l'exemple de política de configuració que hem vist abans.

A més del codi de SELinux, necessitarem alguns paquets de l'espai d'usuari retocats per SELinux per a integrar-los completament en aquest entorn. Existeixen paquets per unes quantes distribucions GNU/Linux, disponibles a partir de http://selinux.sourceforge.net.

Un cop tinguem aquestes aplicacions de més necessàries, haurem d'activar el mòdul de seguretat al kernel. Això ho farem habilitant l'opció de configuració del kernel `CONFIG_SECURITY_SELINUX` que trobarem a *Security Options -> NSA SELinux Support*. El suport per a SELinux només el podem habilitar en forma de mòdul compilat directament al kernel, i un cop instal·lat no podrem carregar-lo ni descarregar-lo. A més del mòdul principal, hi ha algunes opcions de configuració extres, que podem habilitar segons ens convingui. Per últim, disposar de les opcions d'auditoria de SELinux, haurem d'habilitar dues opcions més: la `CONFIG_AUDIT` i la `CONFIG_AUDITSYSCALL`, que proporcionen una crida al sistema per a poder realitzar aquest tipus d'accions d'auditoria.

7.- Seguretat al marge de mòduls del kernel: els *patchs*

Fins ara, hem vist l'aplicació de models de seguretat en forma de mòduls (carregables o complilats) del nucli, i ens hem centrat en el model de seguretat proposat per LSM i els diferents mòduls implementats actualment als kernel Linux. Al llarg del treball, però, hem anat comentant que exisistia una altra manera d'aplicar noves funcionalitats a un kernel Linux, mitjançant *patchs* o pedaços.

En aquest darrer apartat, farem una pinzellada a alguns dels projectes de seguretat existents avui en dia, i que funcionen al marge del *framework* proporcionat per LSM. De fet, hi ha polèmica[1] al respecte, tot i que no hi entrarem, i ens centrarem en una breu enumeració de funcionalitats proporcionades per aquests projectes. Els projectes de seguretat que veurem són són **grsecurity**[2] i **Dazuko**[3]. La presentació d'aquests projectes és purament testimonial, únicament per què en coneguem la seva existència i característiques principals, però no ens hi estendrem com hem fet amb els mòduls que hem vist fins ara, ja que aquests *patchs* no vénen inclosos al kernel i s'escapen dels objectius del treball. No per això, però l'aplicació d'aquests tipus de models de seguretat deixa de tenir validesa, i a vegades els patchs poden ser-nos de gran utilitat. Caldrà tenir-los en compte a l'hora de dissenyar la política de seguretat al nostre sistema.

• Projecte grsecurity:

Grsecurity és un projecte que va nèixer l'any 2001 amb la intenció de fer un *port* de la distribució Openwall[4] al kernel Linux 2.4, ja que aquesta només soportava la branca 2.2. A la documentació de grsecurity es comenta la filosofia que hi ha al darrere, basada en els següents punts:

- La seguretat no es pot solucionar amb una sola capa.
- La seguretat addicional, si no és fàcil d'usar i d'entendre, no serveix per res.
- Hauríem de poder protegir qualsevol mena de programari instal·lat al nostre

1 http://www.grsecurity.net/lsm.php
2 http://www.grsecurity.org
3 http://www.dazuko.org
4 http://www.openwall.com

sistema, no solament el proporcionat per la nostra distribució.

- Els humans som sovint el punt més feble en la seguretat.

Partint d'aquests principis, grsecurity proporciona un munt de funcionalitats de seguretat, de les quals destaquen:

- Prevenció de l'explotació de desbordament de *buffer*: fruit de la col·laboració amb el projecte PaX[1] que, a més d'evitar els *buffer overflow*, proporciona una aleatorització de com es s'escampa per la memòria un procés, que permet evitar atacs orientats a saturar la memòria.

- Control d'accés RBAC

- Protecció del sistema de fitxers: grsecurity afegeix restriccions a `/proc` per tal que els usuaris només puguin veure els processos que estan executant ells. També protegeix el directori /tmp d'explotacions de *tmp-race* i estableix proteccions sobre `chroot(8)` que milloren la seguretat d'aquesta utilitat.

- Mecanismes d'auditoria del kernel per poder resseguir fallades o comportaments no esperats del nostre sistema.

- Protecció d'executables: en la creació dels processos (s'aleatoritza la generació dels identificadors de processos) i definint quins fitxers binaris poden ser executats.

- Seguretat a la xarxa: grsecurity permet l'aleatorització de la pila TCP/IP i decidir quins tipus de *sockets* poden utilitzar els usuaris.

- Suport a `sysctl(8)`: d'aquesta manera podem canviar la configuració de grsecurity en temps d'execució.

El llistat de característiques complet és bastant extens i ens demostra que és un projecte prou potent i que cal tenir en compte com a alternativa a l'hora d'aplicar models de seguretat. Existeix suport per a les versions 2.4 i 2.6 del nucli Linux.

1 http://pax.grsecurity.net

• Projecte Dazuko:

El projecte Dazuko proporciona una interfície, en forma de mòdul del kernel per controlar dispositius, que permet a les aplicacions d'usuari realizar un control d'accés exclusivament sobre fitxers. Va ser desenvolupat originàriament per l'empresa H+BEDV Datentechnik GmbH[1] per tal de permetre als usuaris de fer escaneig de virus, una operació que requereix certs privilegis. Dazuko intercepta les crides per accedir als fitxers i li passa la informació del fitxer a una aplicació d'usuari. Aquesta aplicació pot dir-li al controlador del dispositiu si permet o denega l'accés al recurs, permeten definir les polítiques d'accés a des de fora de l'espai de kernel. La informació del fitxer ve acompanyada de l'event que vol accedir, quin fitxer és l'implicat en l'operació, el tipus d'accés i els identificadors de procés i d'usuari.

La principal aportació de Dazuko, que tot i provenir d'una empresa privada que es dedica a vendre programari està llicenciat com a GPL en la seva versió per plataformes GNU/Linux (per altres plataformes utilitza la llicència BSD), consisteix en donar la oportunitat de desenvolupar aplicacions fora del nucli del kernel que realitzen funcions fins ara reservades a aquest. Proporcionant la interfíce per accedir a totes les funcionalitats del controlador de dispositius permet a desenvolupadors externs crear aplicacions que requereixin un control d'accés a fitxers sense necessitat de tocar el codi del kernel per a res.

A més, un dels altres avantatges de Dazuko és que proporciona interfícies per diferents llenguatges de programació, que inclouen C, Perl, Java i Python, permetent una major flexibilitat als programadors.

Dazuko no proporciona un model de seguretat en si mateix, però permet el desenvolupament d'aplicacions que sí que ajuden a millorar la seguretat dels nostres sistemes. Actualment unes quantes aplicacions[2] l'utilitzen, la majoria d'elles antivirus.

Les plataformes suportades per Dazuko no es limiten a GNU/Linux (per als nuclis compresos entre les versions 2.2 i 2.6), sinó que també accepta Linux/RSBAC[3] i FreeBSD 4/5.

1 http://www.antivir.de
2 http://www.dazuko.org/applications.shtml
3 http://www.rsbac.org

8.- Conclusions

Després de descriure l'escenari a partir de qual hem desenvolupat aquest treball per via de la introducció, els conceptes previs i les novetats introduïdes al kernel Linux 2.6, hem tingut les bases necessàries per centrar-nos en els objectius que ens havíem proposat. Aquests consistien en estudiar les peculiaritats dels models de seguretat incorporats a la versió del kernel Linux.

La solució oferta pel projecte LSM seguint les directrius apuntades per Linus Torvalds, que s'haurien de tenir en compte a l'hora d'incloure una nova estructura de seguretat al kernel, és una bona aproximació. La definició de polítiques de seguretat que s'implementen com a mòdul del kernel, permet situar els desenvolupadors a l'espai del nucli, on podran moure's amb total llibertat per fer les comprovacions que creguin convenients o impedir, si cal, l'accés a recursos sol·licitats.

Els camps de seguretat LSM ens donen la possibilitat de desar i obtenir informació de seguretat de la majoria d'objectes del kernel. Aquests camps han estat afegits a les estructures de dades que representen els processos, els programes, el sistema de fitxers, els inodes, els fitxers, els *buffers* de xarxa, els dispositius de xarxa i aquells relacionats amb la comunicació entre processos, de manera que queden cobertes la majoria de les funcionalitats exigides a un sistema operatiu.

Amb els ganxos a funcions LSM repartits per punts clau del codi del kernel, hem vist com és possible implementar funcions per què siguin cridades quan l'execució del codi passa pel ganxo. Les funcions que s'implementin poden utilitzar llavors els camps de seguretat dels objectes implicats en el moment de la crida i realitzar un control d'accés tant sobre els objectes com sobre els recursos sol·licitats.

També hem pogut comprovar que el moment de la inicialització d'un mòdul de seguretat és un altre dels punts clau, ja que depenent de la política que es vulgui utilitzar, hem de garantir que tenim el sistema en l'estat que volíem per a començar a aplicar aquesta política. El cas extrem de SELinux, que divideix el procés de la càrrega del mòdul en sis etapes, ens ha servit d'exemple per veure la importància d'aquest moment.

Pel que fa als models de seguretat objecte d'estudi, els tres primers que hem vist a la secció 4.2 del treball, ens han servit per veure com s'integra un mòdul de seguretat a l'entorn LSM. Dels tres, l'únic que realment implementa un model de seguretat és el BSD Secure Levels, tot i que podent utilitzar les Linux Capabilities que ens proporcionen més refinament en el control d'accés, potser deixa de tenir sentit la utilització d'aquests nivells

de seguretat.

Les Linux Capabilities han estat el primer model de seguretat basat en LSM amb cara i ulls que hem vist. Proporcionen una divisió dels privilegis que afegeixen una granularitat molt útil en el control d'accés a les operacions privilegiades. Cal remarcar que tant les Linux Capabilities com SELinux ens seran útils en màquines en producció funcionant a ple rendiment, ja que en sistemes domèstics, les incomoditats que comporta un control tan rígid sobre els recursos poden arribar a desesperar l'usuari.

Sigui com sigui, amb les Linux Capabilities podem garantir un nivell de seguretat prou elevat d'una manera ràpida i senzilla si sabem que el sistema informàtic on s'aplica no ha de patir modificacions de maquinari o de configuració que requereixin la intervenció d'un usuari administrador.

Tot i així, com que aquestes capacitats s'apliquen únicament sobre processos i fitxers executables, la falta d'un control d'accés pels fitxers no executables és una de les mancances de les Linux Capabilities recriminades a aquest model.

L'altre model de seguretat LSM més robust que hem vist és el NSA SELinux. Aquesta modificació del kernel tenia com a objectiu demostrar la validesa dels control d'accés de tipus MAC. Com hem vist, aquest tipus de control d'accés és molt més segur que els emprats tradicionalment als sistemes UNIX, de tipus DAC.

Per tal d'aconseguir els seus objectius, SELinux associa un domini a cada procés i un tipus a cada objecte del kernel i, mitjançant unes taules indexades, pot decidir en tot moment si un procés pertanyent a un domini pot accedir a un objecte d'un tipus concret. Per reforçar més la seguretat, també es defineixen rols d'usuaris, que es defineixen com a usuaris normals o com a administradors. D'aquesta manera s'elimina la dependència d'un compte de `root` per a l'administració del sistem i també s'eliminen els vestigis d'un control d'accés tipus DAC.

A més, SELinux defineix una quantitat considerable de ganxos a funcions LSM, de manera que permet arribar a controlar el més mínim detall de les operacions que està realitzant el nucli en tot moment.

Realment el model SELinux és el més complet dels vistos, però també el més complex, tant d'entendre com de configurar, ja que permet un nivell de detall tant elevat que és fàcil perdre's durant la seva activació. Més endavant, quan hem fet un repàs als *patchs* del kernel usats per aplicar altres models de seguretat, la màxima del projecte grsecurity: “La seguretat addicional, si no és fàcil d'usar i d'entendre, no serveix per res”, ha pres tot el seu sentit.

Podem considerar assolits els objectius marcats a l'hora de presentar la proposta d'aquest treball, ja que hem vist quins eren els models de seguretat proporcionats per la versió del kernel 2.6 i la seva implementació. A més, durant el procés, hem hagut de consultar de manera freqüent les fonts del kernel, fet que ens ha ajudat a entendre una mica millor quina és la seva organització i quins són els actors principals que permeten el funcionament del sistema operatiu GNU/Linux.

Potser ens ha quedat per cobrir, d'una manera més extensa, els models de seguretat que no van ser incorporats a la branca oficial del kernel, com són els dos projectes que hem vist, grsecurity i Dazuko. Hem pogut veure que el fet de ser implementats en forma de *patch*, deixant de banda la limitació de no poder disposar de codi que pot ser carregat de manera dinàmica com pot fer-se amb els mòduls del kernel, no els converteix en menys vàlids. L'estudi d'aquest tipus de models queda obert, ja que hem vist que són prou interessants com per que valgui la pena dedicar-hi un altre treball.

9.- Bibliografia

Bibliografia impresa:

- Carretero, J.; García, F.; Miguel, P. de; Pérez, F.; 2001 : *Sistemas Opertivos: Una visión aplicada*, Madrid, Ed. McGraw Hill.
- Kernighan, B. W. ; Pike, R.; 1987 : *El entorno de programación UNIX*, Mèxic, Ed. Prentice Hall.
- Welsh, M. ; Kalle, M. ; Kaufman, L. ; 2000 : *Guía de referencia y aprendizaje Linux*, Madrid, Ed. Anaya Multimedia.
- Sánchez, S. ; 1999 : *UNIX y Linux. Guía práctica*, Madrid, Ed. Ra-ma.
- Hontañón, R. J. ; 2001 : *Linux Security*, Nova Delhi, Ed. BPB Publications.
- López, A. ; Novo, A. ; 1999 : *Protocolos de Internet. Diseño e implementación en sistemas UNIX*, Madrid, Ed. Ra-ma.
- Fernández, D. ; Sánchez, J. ; Cantallops, G. (juliol 2002): *"Especial Seguridad Linux"*. *Sólo Programadores*. Madrid, Ed. Revistas Profesionales.

Recursos electrònics:

- Manual del programador Linux (pàgines `man`)
- `/usr/src/linux/Documentation/` : Directori amb la documentació del codi font del nucli
- Codi font del nucli 2.6.12.1 i anteriors
  [consulta 09/2005] accessible a:
  http://www.kernel.org

- Fernández-Sanguino Peña, J. ; 2004 : *Manual de Seguridad de Debian*
  [consulta 04/2005] accessible a:
  http://www.debian.org/doc/manuals/securing-debian-howto/index.es.html

- Wreski, D. ; agost 1998 : *Linux Security Administrator's Guide*
  [consulta 04/2005] accessible a:
  http://www.linuxsecurity.com/docs/SecurityAdminGuide/SecurityAdminGuide.html

- Fenz, K ; Wreski, D. ; gener 2004 : *Linux Security HOWTO*
  [consulta 04/2005] accessible a:
  http://en.tldp.org/HOWTO/Security-HOWTO/

- Wolf, G. ; 2003 : *Evolución de los esquemas de seguridad extendidos en Unix*
  [consulta 06/2005] accessible a:
  http://www.gwolf.org/seguridad/evol_esq_seg/evol_esq_seg.html

- Bagwill, R. ; *et al.* ; 1994: *Security in Open Systems*
  [consulta 04/2005] accessible a:
  http://csrc.nist.gov/publications/nistpubs/800-7/main.html

- Rusling, D. A. ; 1999 : *The Linux Kernel*
  [consulta 04/2005] accessible a:
  http://en.tldp.org/LDP/tlk/tlk.html

- Bowman, I. T. ; desembre 1998 : *Conceptual Architecture of the Linux Kernel*
  [consulta 07/2005] accessible a:
  http://plg.uwaterloo.ca/~itbowman/CS746G/a1/

- Bowman, I. T. ; Siddiqi, S. ; Tanuan, M. C. ; octubre 1998 : *Concrete Architecture of the Linux Kernel*
  [consulta 07/2005] accessible a:
  http://plg.uwaterloo.ca/~itbowman/CS746G/a2/

- Gooch, R. ; setembre 2002 : *I/O Event Handling Under Linux*
  [consulta 05/2005] accessible a:
  http://www.atnf.csiro.au/people/rgooch/linux/docs/io-events.html

- Salzman, P. J. ; Burian, M. ; Pomerantz, O. ; maig 2005 : *The Linux Kernel Module Programming Guide*
  [consulta 03/2005] accessible a:
  http://www.tldp.org/LDP/lkmpg/2.6/html/index.html

- Ebling, A. ; març 2002 : *Kernel Hacking HOWTO*
  [consulta 04/2005] accessible a:
  http://www.kernelhacking.org/docs/kernelhacking-HOWTO/

- Maeda, T. ; 2002 : *Kernel Mode Linux: Execute user processes in kernel mode*
  [consulta 03/2005] accessible a:
  http://web.yl.is.s.u-tokyo.ac.jp/~tosh/kml/

- `pragmatic` / THC ; març 1999 : *(nearly) Complete Linux Loadable Kernel Modules*
  [consulta 05/2005] accessible a:
  http://packetstormsecurity.org/docs/hack/LKM_HACKING.html

- `plaguez`; gener de 1998 : *"Weakening the Linux Kernel"*. *Phrack* núm 52
  [consulta 05/2005] accessible a:
  http://www.phrack.org/phrack/52/P52-18

- Wheeler, D. ; 2001: *Secure Programming for Linux and Unix HOWTO*
  [consulta 04/2005] accessible a:
  http://cs.mipt.ru/docs/comp/eng/os/linux/howto/secure_programs/

- Coar, K. ; febrer 2001 : *Mandatory Versus Discretionary Access Control*
  [consulta 06/2005] accessible a:
  http://www.linuxplanet.com/linuxplanet/tutorials/1527/2/

- Tobotras, B. ; 1999 : *Linux Capabilities FAQ 0.2*
  [consulta 06/2005] accessible a:
  http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

- Bacarella, M. ; 2002 : *Taking Advantage of Linux Capabilities*. Linux Journal
  [consulta 06/2005] accessible a:
  http://www.linuxjournal.com/article/5737

- Exequiel, G. ; 2003 : *La implementación de POSIX Capabilities en Linux*.
  [consulta 06/2005] accessible a:
  http://www.psicofxp.com/forums/showthread.php?t=112458

- Miller, M. ; Yee, K. ; Shapiro, J. ; *Capability Myths Demolished*
  [consulta 06/2005] accessible a:
  http://zesty.ca/capmyths/

- Trümper, W. ; 1999 : *Summary about Posix.1e*
  [consulta 06/2005] accessible a:
  http://wt.xpilot.org/publications/posix.1e/

- Martínez, J. ; 2004; *Asegurando Linux: Las Linux Capabilities*
  [consulta 06/2005] accessible a:
  http://www.esdebian.org/staticpages/index.php?page=20040715023347827

- Martínez, J. ; 2004; *User Mode Linux + Linux capabilities. Aplicación a servidores* [consulta 06/2005] accessible a: http://www.esdebian.org/staticpages/index.php?page=20041021030032927

- Rauch, J. ; 2000: *Introduction to Linux Capabilities and ACL's* [consulta 06/2005] accessible a: http://www.securityfocus.com/infocus/1400

- killa.net; 1999: Implementing POSIX 1003.1e capabilities under Linux [consulta 06/2005] accessible a: http://killa.net/infosec/caps/

- Williamson, R. ; març 2004: *Networking improvements in the 2.6 kernel* [consulta 08/2005] accessible a: http://www-128.ibm.com/developerworks/linux/library/l-net26.html

- Santhanam, A. ; setembre 2003 : *Towards Linux 2.6* [consulta 08/2005] accessible a: http://www-128.ibm.com/developerworks/linux/library/l-inside.html

- Peredo, O. ; 2003: *Comenzando con el kernel 2.6* [consulta 08/2005] accessible a: http://red.uninet.edu/josu/n1/operedo.html

- Pravenich, J. ; 2003: *El Maravilloso Mundo de Linux 2.6* [consulta 08/2005] accessible a: http://www.escomposlinux.org/wwol26/wwol26.html

- Jones, D. ; 2003: *El documento de post-halloween* [consulta 08/2005] accessible a: http://www.terra.es/personal/diegocg/post-halloween-2.6.es.txt

- Open Source Development Labs Linux, Inc; 2003 : *Process Scheduler Improvements* [consulta 08/2005] accessible a: http://developer.osdl.org/craiger/hackbench/index.html

- Venezia, P. ; 2004 : *Linux v2.6 scales the enterprise* [consulta 08/2005] accessible a: http://www.infoworld.com/infoworld/article/04/01/30/05FElinux_1.html

- Wright, C. ; *et al.* ; 2003: *General Security Support for the Linux Kernel* [consulta 03/2005] accessible a: http://lsm.immunix.org/docs/lsm-usenix-2002/html/

- Smalley, S; Fraser, T. ; Vance, C; *General Security Hooks for Linux*
  [consulta 08/2005] accessible a:
  http://lsm.immunix.org/docs/overview/linuxsecuritymodule.html

- Smalley, S. ; Vance, C. ; Salamon, W; 2005 : *Implementing SELinux as a Linux Security Module*
  [consulta 08/2005] accessible a:
  http://www.nsa.gov/selinux/papers/module-abs.cfm

- Smalley, S. ; 2005 : *Configuring the SELinux Policy*
  [consulta 08/2005] accessible a:
  http://www.nsa.gov/selinux/papers/policy2-abs.cfm

- Smalley, S; Fraser, T. ; 2001: *A Security Policy Configuration for the Security-Enhanced Linux*
  [consulta 09/2005] accessible a:
  http://www.nsa.gov/selinux/papers/policy-abs.cfm

- Halcrow, M. ; setembre 2004 : *"Using the BSD Secure Levels LSM"*. *Sys Admin* v13, i09
  [consulta 07/2005] accessible a:
  http://www.samag.com/documents/s=9304/sam0409a/0409a.htm

- Kroah, G; 2002 : *Using the Kernel Security Module Interface*
  [consulta 07/2005] accessible a:
  http://www.linuxjournal.com/article/6279

- Grünbacher, A. ; 2003: Extended attributes and ACLs for Linux
  [consulta 09/2005] accessible a:
  http://acl.bestbits.at/

- Grünbacher, A. ; 2003: POSIX Access Control Lists on Linux
  [consulta 09/2005] accessible a:
  http://www.suse.de/~agruen/acl/linux-acls/online/

- Spengler, B. ; 2002 : *Detection, prevention and containment: a study of grsecurity*
  [consulta 09/2005] accessible a:
  http://www.grsecurity.net/grsecurity-slide_files/frame.htm

- grsecurity: *Grsecurity Quick-Start Guide*
  [consulta 09/2005] accessible a:
  http://www.grsecurity.net/quickstart.pdf

- *Dazuko*
  [consulta 09/2005] accessible a:
  http://www.dazuko.org/

- Altres recursos:

- *The Community's Center for Security*
  [consulta 04/2005] accessible a:
  http://www.linuxsecurity.com/component/option,com_weblinks/catid,155/Itemid,134/

- *Linux HeadQuarters : Kernel Module Programming*
  [consulta 04/2005] accessible a:
  http://www.linuxhq.com/lkprogram.html

- *Capability Systems References*
  [consulta 06/2005] accessible a:
  http://www.zilium.de/joerg/capbib.html

- *Linux Kernel 2.6 Status*
  [consulta 08/2005] accessible a:
  http://www.kernelnewbies.org/status/

- Solworth, J. ; 2005 : KernelSec: *Kernel Protection Model to Implement Security Policies*
  [consulta 05/2005] accessible a:
  http://parsys.cs.uic.edu/~solworth/kernelSec.html

- *SecDev.org*
  [consulta 04/2005] accessible a:
  http://www.secdev.org/

- *LIDS Project - Secure Linux System*
  [consulta 04/2005] accessible a:
  http://lids.org/

- The User-mode Linux Kernel Home Page
  [consulta 06/2005] accessible a:
  http://user-mode-linux.sourceforge.net/

- *Documentos e-ghost*
  [consulta 06/2005] accessible a:
  http://www.e-ghost.deusto.es/phpwiki/index.php/DocuMentos

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

UNIVERSITAT DE BARCELONA

Treball fi de carrera presentat el dia de de 200
a la Facultat de Matemàtiques de la Universitat de Barcelona,
amb el següent tribunal:

Dr. President

Dr. Vocal 1

Dr. Secretari

Amb la qualificació de: