امنیت هسته Linux

.

فهرست

مقدمه

امروزه استفاده از شبکه و اینترنت برای ارائهٔ سرویس‌های مختلف در حد بسیار زیادی رشد کرده‌است. برای قابل استفاده بودن سرویس‌هایی که روی اینترنت ارائه می‌شوند برقراری یک امنیت نسبی لازم به‌نظر می‌رسد. البته در برخی حیطه‌های خاص مانند تجارت الکترونیک و بانکداری اینترنتی لزوم برقراری امنیت کامل که اطمینان خاطر مشتریان را فراهم سازد بیشتر نمود پیدا می‌کند. با توسعهٔ این نوع سیستم‌ها نیازهای امنیتی بیشتری احساس می‌شود که باید از طرف برنامه‌های کاربردی مربوطه و نیز ازجانب سیستم‌عامل‌ها پشتیبانی شوند.

امروزه Linux به‌عنوان یک سیستم‌عامل قدرتمند شناخته شده‌است و استفاده از آن برای راه‌اندازی سرویس‌های مبتنی بر اینترنت روزبه‌روز بیشتر می‌شود. در این متن به بررسی امکانات امنیتی که در هستهٔ Linux وجود دارد پرداخته می‌شود. در ابتدا مفاهیمی مانند ساختار امنیتی flask و تکنولوژی‌های TE و RBAC بیان می‌شود، سپس درمورد زیرساخت امنیتی LSM به همراه ماجول‌های امنیتی Capability، Openwall، LIDS، DTE و SELinux توضیح داده می‌شود.

۱ مفاهیم امنیتی

در این قسمت از متن برخی تکنولوژی‌های متداول در زمینهٔ امنیت معرفی می‌شوند که دانستن برای فهم نحوهٔ عملکرد ماجول‌های امنیتی ذکرشده در قسمت بعدی متن ضروری می‌باشد. مواردی که در این قسمت به آنها اشاره می‌شود عبارتند از:

- ساختار امنیتی flask
- اِعمال تایپ یا TE[1] (DTE[2])
- کنترل دسترسی بر مبنای نقش یا RBAC[3]

۱-۱ ساختار امنیتی flask

محیط‌های مختلف عملیاتی و محاسباتی نیازهای امنیتی متفاوتی دارند، به همین علت هر سیستم خاص از یک مکانیزم و سیاست امنیتی مخصوص به خود استفاده می‌کند. همین امر لزوم بودن سیاست‌های مختلف و حتی انواع مختلف سیاست‌های امنیتی را روشن می‌کند. یک سیستم امنیتی باید به اندازهٔ کافی انعطاف‌پذیر باشد که بتواند این طیف گستردهٔ مکانیزم‌های امنیتی را پشتیبانی کند. انعطاف‌پذیری به‌معنای امکان استفاده از چند سیاست امنیتی مختلف نمی‌باشد. لازمهٔ این انعطاف پذیری دارا بودن سه فاکتور زیر است:

- کنترل صدور مجوزهای دسترسی
- اعمال مجوزهای دسترسی در سطوح پایین
- امکان بازپس‌گیری مجوزهای صادرشده

۱-۱-۱ flask به‌عنوان یک ساختار امنیتی انعطاف‌پذیر

کامپیوتر را به‌عنوان یک ماشین حالت در نظر بگیرید که با انجام دادن اعمال اتمیک از یک وضعیت به وضعیت دیگر تغییر حالت می‌دهد. این سیستم وقتی می‌تواند دارای امنیت انعطاف‌پذیر باشد که سیاست امنیتی بتواند در هر عملی که توسط سیستم انجام می‌شود مداخله نماید؛ یعنی به یک عمل اجازهٔ انجام شدن را بدهد، مقابل انجام آن را بگیرد، یا اینکه یک‌سری

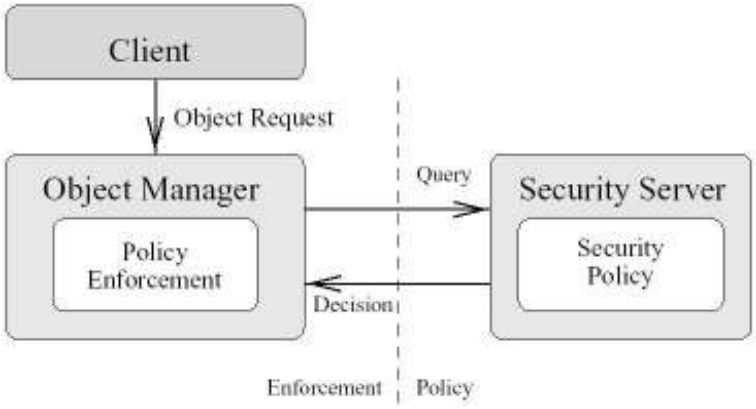
[1] Type Enforcement
[2] Domain and Type Enforcement
[3] Role-Based Access Control

عملیات دیگر را در اثنای کار داخل سیستم اضافه کند. در چنین سیستمی تصمیمات امنیتی با توجه به وضعیت کل سیستم اتخاذ می‌شود. ازآنجاکه مداخله در هر عملی که توسط سیستم انجام می‌شود امکان‌پذیر است، لذا بازپس‌گیری مجوزهای صادرشده به‌سادگی انجام می‌گیرد. معمولاً از وضعیت کل سیستم برای تصمیم‌گیری درمورد تمام عملکردهای ممکن استفاده نمی‌شود. راه واقع‌بینانه‌تر این است که قسمت‌هایی از وضعیت سیستم که در اتخاذ تصمیمات امنیتی موثرند و عملکردهایی از سیستم که برروی این قسمت از سیستم تاثیر می‌گذارند یا از آن تاثیر می‌پذیرند شناسایی گردند. این کار باعث می‌شود که بخشی از اعمال سیستم از کنترل سیاست امنیتی خارج شود و بعضی از حالت‌های سیستم خارج از محدودهٔ تعریف‌شده توسط سیاست امنیتی وجود داشته باشد. در این حالت این تضمین باید وجود داشته باشد که تمام مجوزهای صادرشده درصورت تغییر سیاست امنیتی قابل بازپس‌گیری باشد.

ساختار امنیتی flask نمونه‌ای از یک سیستم انعطاف‌پذیر امنیتی می‌باشد. flask محصول کار مشترک محققین سازمان امنیت ملی، دانشگاه Utah و شرکت SecureComputing می‌باشد. در این سیستم صدور مجوزهای دسترسی مطابق با سیاست امنیتی می‌باشد، یعنی هرگونه تصمیم امنیتی اول با سیاست امنیتی چک می‌شود. البته این کار تغییر محسوسی در سرعت عملکرد سیستم ندارد چون تصمیمات امنیتی اتخاذشده از طریق مکانیزم‌های مختلف caching برای استفاده‌های بعدی ذخیره می‌شوند. اعمال مجوزها از طریق مکانیزم‌هایی که مستقیماً در اجزای سرویس‌دهندهٔ سیستم قرار داده شده‌اند کنترل می‌شوند، به‌همین دلیل این مجوزها در پایین‌ترین سطوح نیز قابل اعمال می‌باشند. بازپس‌گیری مجوزهای صادرشده نیز توسط پروتکل‌های خاصی که برای این کار وضع شده‌اند صورت می‌گیرد.

۲-۱-۱ عملکرد داخلی flask

flask، همان‌طورکه در شکل ۱ نشان داده شده‌است، دارای دو بخش کاملاً مجزا می‌باشد: سرور امنیتی که سیاست امنیتی را تحلیل کرده و مجوزهای امنیتی را براساس آن صادر می‌کند، و یک‌سری object manager که تصمیمات و مجوزهای امنیتی صادرشده را در سطوح پایین برروی بخش‌های مختلف سیستم اعمال می‌کنند. هدف اصلی چنین ساختاری بالا بردن انعطاف‌پذیری سیاست امنیتی می‌باشد. برای تغییر سیاست امنیتی فقط کد سرور امنیتی تغیر می‌کند و object managerها بدون تغییر باقی می‌مانند. object managerها همواره یک دید ثابت نسبت به تصمیمات و مجوزهای صادرشده دارند، بدون توجه به اینکه این تصمیمات چگونه گرفته می‌شوند یا اینکه چگونه با مرور زمان تغییر می‌کنند.



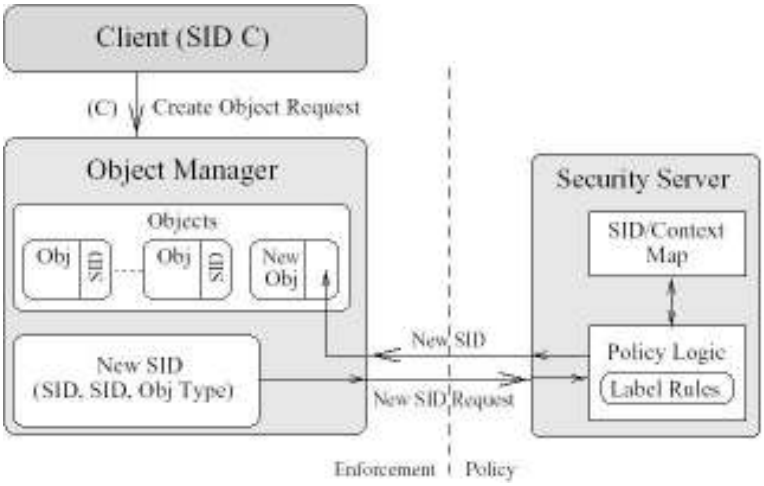
شکل ۱ – ساختار داخلی flask

تمام اشیائی که توسط سرور امنیتی کنترل می‌شوند توسط یک‌سری از خصوصیات امنیتی که به مجموعهٔ آنها context امنیتی گفته می‌شود برچسب[۴] زده می‌شوند. context امنیتی شامل تمام خصوصیات امنیتی می‌باشد که به یک شیء خاص نسبت داده شده‌اند. ازآنجاکه محتوا و فرمت context امنیتی به مدل امنیتی مورد استفاده بستگی دارد، لذا context امنیتی فقط توسط سرور امنیتی تفسیر می‌شود. هر context امنیتی یک شمارهٔ SID[۵] متناظر دارد که object managerها از آن استفاده می‌کنند و توسط سرور امنیتی تبدیل به context امنیتی می‌شوند. این شماره‌ها توسط سرور امنیتی هنگام شروع کار سیستم به اشیاء نسبت داده می‌شوند و در هر مرتبه اجرای سیستم تغییر می‌کنند. استفاده از SID باعث می‌شود object managerها بی‌نیاز از دانستن محتوا و حتی فرمت context امنیتی باشند. برای مدیریت فایل‌ها از PSID[۶] به‌جای SID استفاده می‌شود که هنگام اجرای مجدد سیستم تغییر نمی‌کنند و همراه فایل‌سیستم ذخیره می‌شوند؛ همین باعث می‌شود که درصورت انتقال فایل‌سیستم اطلاعات مربوط به خصوصیات امنیتی فایل‌ها از بین نرود.

تصمیماتی که از جانب سرور امنیتی گرفته می‌شود به سه قسمت تقسیم می‌شوند: برچسب زدن، دسترسی، و polyinstantiation. تمام اجزای سیستم که توسط سیاست امنیتی کنترل می‌شوند باید دارای برچسب‌های امنیتی باشند. برای هر شیء جدیدی که ایجاد می‌شود یک context امنیتی محاسبه می‌شود و یک SID به آن نسبت داده می‌شود که context امنیتی مربوطه را مشخص می‌کند. context امنیتی یک شیء معمولاً وابسته به سرویس‌گیر متقاضی ایجاد شیء و محیطی که شیء در آن ایجاد شده می‌باشد. برای مثال context امنیتی یک فایل

[4] label
[5] Security Identifier
[6] Persistent Security Identifier

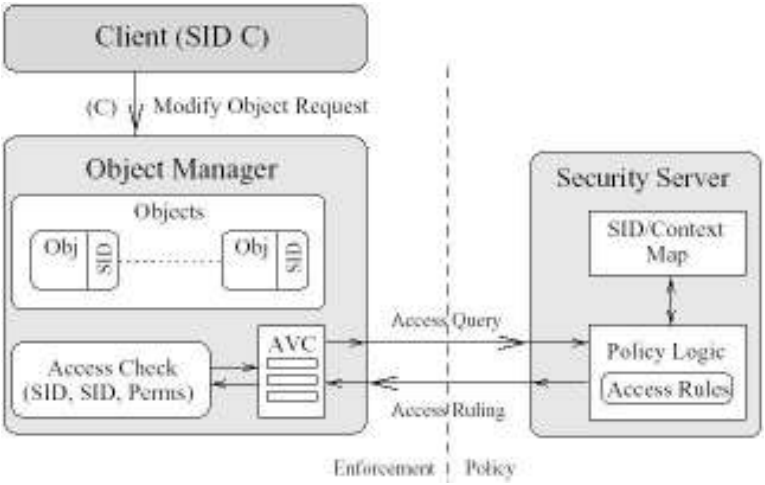
جدید وابسته به context امنیتی پروسهٔ ایجادکنندهٔ فایل و دایرکتوری که فایل در آن ایجاد می‌شود می‌باشد. ازآنجاکه محاسبهٔ context امنیتی یک شیء جدید وابسته به سیاست امنیتی می‌باشد، لذا این عمل توسط سرور امنیتی انجام می‌گردد. فرایند تخصیص context امنیتی برای یک شیء جدید در شکل ۲ نشان داده شده‌است.



شکل ۲ – تخصیص context امنیتی به یک شیء جدید

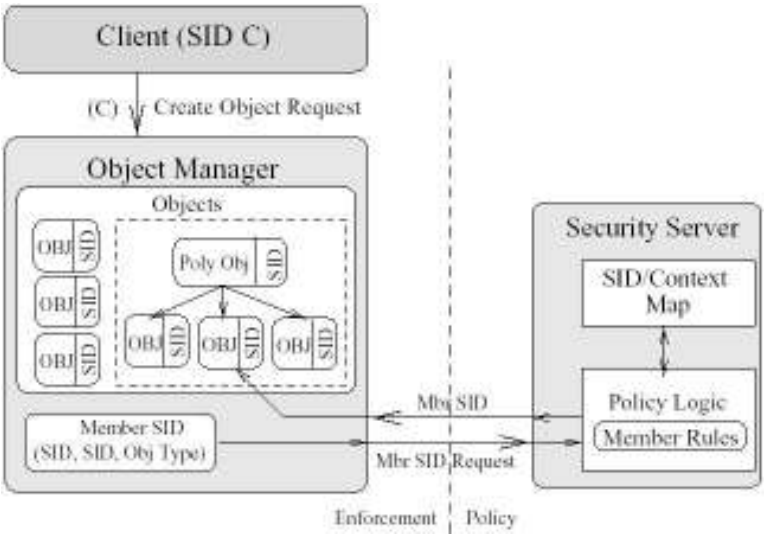
در ساده‌ترین حالت object manager می‌تواند در مواقعی که نیاز به تصمیمات امنیتی دارد یک تقاضا برای سرور امنیتی بفرستد. این کار سرعت عملکرد سیستم تا حد قابل‌توجهی پایین می‌آورد. برای بهبود سرعت عملکرد سیستم از یک مکانیزم caching به‌نام AVC[۷] استفاده می‌کند. AVC یک مکانیزم مشترک بین تمام object managerها می‌باشد. در مواقعی که نیاز به یک تصمیم امنیتی باشد یک تقاضا برای سرور فرستاده می‌شود و سرور در پاسخ دسترسی درخواست شده و یک‌سری دسترسی‌های مشابه را در یک ساختار به‌نام Access Vector برای object manager می‌فرستد. تمام این دسترسی‌ها در AVC برای استفاده‌های بعدی ذخیره می‌شوند. نحوهٔ عملکرد AVC در شکل شمارهٔ ۳ نشان داده شده‌است.

[7] Access Vector Cache



شکل ۳ – نحوهٔ عملکرد AVC

برای استفاده از اشیاء مشترک در سیستم، flask از polyinstatiation یا چندنمونه‌سازی استفاده می‌کند. در این روش یک شیء (مثلاً دایرکتوری tmp/) به چند نمونهٔ مختلف تقسیم می‌شود که از هر نمونه یک گروه از سرویس‌گیرها می‌توانند استفاده کنند. از آنجا که تمام نمونه‌های شیء موردنظر با یک SID مشخص می‌شود برای مشخص کردن نمونهٔ موردنظر از آن شیء از تصمیمات polyinstatiation استفاده می‌شود. نحوهٔ عملکرد سیستم polyinstantiation در شکل ۴ نشان داده شده‌است.



شکل ۴ – نحوهٔ عملکرد polyinstatiation

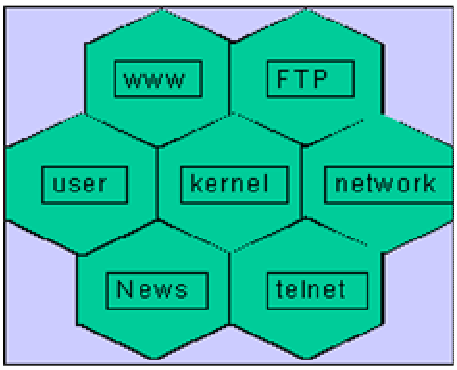
برای انعطاف‌پذیر بودن ساختار امنیتی سیستم، پس از بروز تغییر در سیاست امنیتی باید این تغییر را رفتار object managerها نشان دهد و این کار در طول یک مدت زمان معقول انجام شود. برای این منظور به محض تغییر کردن سیاست امنیتی، تغییرات ایجادشده به تمام object

managerهایی که از قسمت‌های تغییریافتهٔ سیاست امنیتی استفاده کرده‌اند اعلام می‌شود. در ادامه object managerها نیز در وضعیت داخلی خود و AVC تغییرات لازم را اعمال می‌کنند. درنهایت object managerها انجام شدن تغییرات را به سرور امنیتی اعلام می‌کنند. پس از این‌که تمام object managerها این کار را انجام دادند، تغییر سیاست امنیتی کامل می‌شود.

۲-۱ اِعمال تایپ یا TE (DTE)

تکنولوژی اعمال تایپ توسط شرکت Secure Computing ابداع شد و هدف آن فراهم آوردن یک محیط امن برای سیستم‌های محاسباتی بود به‌نحوی که در ردهٔ A1 از رده‌بندی سیستم‌های امنیتی قرار بگیرند. در TE به هر پروسه یک برچسب domain و به هر شیء (فایل، دایرکتوری، سوکت و ...) یک برچسب تایپ نسبت داده می‌شود. برچسب domain عملکرد یک پروسه و برچسب تایپ نیز هدف یک شیء را نشان می‌دهد. سپس قوانین کنترل دسترسی طوری بین پروسه‌ها و اشیاء سیستم برقرار می‌شود که پروسه‌ها بتوانند عملکرد خود را انجام دهند و اشیاء نیز اهداف خود را برآورده کنند. تمام پروسه‌هایی که در یک domain قرار دارند، همچنین تمام اشیائی که دارای یک تایپ می‌باشند خصوصیات امنیتی یکسانی دارند. یک‌سری جداول دسترسی مشخص می‌کند که چگونه domainها می‌توانند باهم ارتباط داشته باشند یا به تایپ‌ها دسترسی پیدا کنند. دسترسی به تایپ‌ها می‌تواند شامل اجازه‌های خواندن، نوشتن، اجرا، ایجاد، و ... باشد. ارتباط domainها با یکدیگر شامل فرستادن سیگنال و انتقال از یک domain به یک domain دیگر[۸] می‌باشد. هر پروسه در هر لحظه می‌تواند عضو یک domain خاص باشد و انتقال از یک domain به یک domain دیگر ازطریق اجرای یک‌سری برنامهٔ اجرایی خاص که مدخل[۹] نام دارند امکان‌پذیر است. هر کاربر برحسب نقشی که در سیستم دارد و نیز براساس سیاست‌های امنیتی می‌تواند وارد domainهای خاصی شود.

[8] Transision
[9] Entry point



شکل ۵ – شمای کلی TE

شکل ۵ یک شمای کلی از تکنولوژی TE را نشان می‌دهد. این تکنولوژی کاربردهای مختلف را از یکدیگر و همچنین از اجزای خود سیستم‌عامل جدا می‌کند. کنترل‌هایی که در این سیستم برروی دسترسی‌ها وجود دارد کاملاً مقابل حملات stack overrun را می‌گیرد. طبق آمار CERT[10] نزدیک به ۷۰٪ حملات امنیتی از نوع stack overrun می‌باشند. ازآنجاکه هر زیرسیستم کاربردی نیازمندی‌های امنیتی خاص خود را دارد از یک‌سری domainها و تایپ‌های خاص استفاده می‌کند. چون هر پروسه که در یک domain اجرا می‌شود با یک کاربرد خاص سروکار دارد، لذا هر کاربرد در یک محدودهٔ خاصی از domainها اجرا می‌شود. با این اوصاف سرویس‌ها و کاربردهای اینترنت هرکدام در یک ناحیهٔ خاص قرار می‌گیرند. این سرویس‌ها نمی‌توانند به اطلاعات نواحی دیگر دسترسی پیدا کنند مگر این‌که در قوانین انتقال domain صراحتاً ذکر شده باشد. البته مدخل‌هایی که انتقال domain ازطریق آنها انجام می‌شود به‌شدت قابل کنترل می‌باشند. این کار مانع از این می‌شود که یک مهاجم بتواند از طریق یک سرویس خاص به بقیهٔ سیستم صدمه‌ای بزند.

۳-۱ کنترل دسترسی بر مبنای نقش یا RBAC

تحقیقات اخیر برروی تکنولوژی کنترل دسترسی‌ها که توسط وزارت دفاع ایالات متحده انجام گردید منجر به پدیدآمدن دو نوع مختلف کنترل دسترسی‌ها گردید: کنترل اختیاری دسترسی‌ها یا DAC[11] و کنترل اجباری دسترسی‌ها یا MAC[12]. در روش DAC کنترل دسترسی‌ها به اختیار کاربران کنترل می‌شود و کاربران اجازه دارند روی اشیائی که تحت کنترل‌شان می‌باشد دسترسی‌های دلخواه خود را تعریف کنند. در این حالت کاربران مالک اشیاء تحت کنترل خود

[10] Computer Emergency Response Team
[11] Discretionary Access Control
[12] Mandatory Access Control

می‌باشند. در بسیاری از شرایط کاربران مالک اشیاء تحت کنترل خود نمی‌باشند. درحقیقت یک سازمان مالک اطلاعات می‌باشد و کاربران فقط اجازهٔ دسترسی به این اطلاعات را دارند. در روش MAC دسترسی‌ها توسط سازمان کنترل می‌شود و معمولاً براساس عملکرد کاربران این کار انجام می‌شود نه مالکیت اطلاعات.

در RBAC مجوزهای دسترسی براساس نقش‌هایی که کاربران به‌عنوان اجزای یک سازمان دارند صادر می‌شوند. مجوزهای دسترسی برحسب نقش‌ها طبقه‌بندی می‌شوند و کاربران به نسبت نقشی که به خود می‌گیرند از مجوزهای مربوطه استفاده می‌کنند. برای مثال در یک سیستم بیمارستان کسی که در نقش دکتر قرار دارد می‌تواند دارو تجویز کند و کسی که نقش مسئول داروخانه را دارد فقط می‌تواند طبق نسخهٔ تجویزشده دارو تحویل دهد و اجازهٔ تجویز دارو را ندارد. استفاده از نقش‌ها یک راه موثر برای توسعه و پیاده‌سازی قوانین و سیاست‌ها امنیتی سازمان و نیز ساده‌تر کردن فرایند مدیریت امنیتی می‌باشد.

یکی از خصوصیات تکنولوژی RBAC این است که به هر کاربر همان دسترسی‌هایی اعطا می‌شود که برای انجام دادن کار خود به آنها نیاز دارد. کاربران برحسب مسئولیت‌هایی که دارند به عضویت نقش‌های مختلف درمی‌آیند. اعمالی که کاربران می‌توانند انجام دهند ازطریق این نقش‌ها کنترل می‌شود. به‌محض تغییر عملکرد کاربران عضویت در نقش‌ها به‌سادگی قابل تغییر است. ازطرف دیگر با تغییر سیاست‌های سازمان مجوزهای هر نقش نیز قابل تغییر است. این کار مدیریت دسترسی‌ها را ساده‌تر می‌کند. تغییرات دلخواه به‌سادگی برروی نقش‌ها اعمال می‌شود بدون اینکه نیازی به تغییر دسترسی‌های تک‌تک کاربران باشد.

در بعضی مواقع نقش‌های مختلف نقاط اشتراک در عملکرد خود دارند، یعنی ممکن است کاربرانی که به نقش‌های مختلف تعلق دارند بتوانند اعمال مشترکی انجام دهند. در این شرایط مشخص کردن این عملکردها برای تمامی نقش‌ها مدیریت آنها را دشوار می‌کند. چنین مواردی در سلسله مراتب سازمانی به کرات دیده می‌شود. کاربری که دارای مقام بالاتری می‌باشد دارای تمام مجوزهای زیردستیان خود نیز می‌باشد. برای فراهم کردن ساختار طبیعی سازمان از سلسله مراتب نقش‌ها استفاده شود. در این سلسله مراتب یک نقش می‌تواند علاوه‌بر دسترسی‌های خود دسترسی‌های یک نقش دیگر را نیز به‌طور ضمنی دارا باشد (ارث ببرد).

قوانین کلی که بر ساختار RBAC حاکم می‌باشند ازقرار زیر می‌باشد:

- یک کاربر می‌تواند عضو چند نقش مختلف باشد.
- یک کاربر در هر لحظه فقط می‌تواند از یک نقش استفاده کند.

- یک نقش می‌تواند اجازهٔ انتقال به یک نقش دیگر را داشته باشد که این انتقال توسط کاربران انجام می‌شود.

۲ ماجول‌های امنیتی

در مارس ۲۰۰۱ سازمان امنیت ملی ایالات متحده (NSA)[۱۳] یک نسخهٔ خاص از Linux به‌نام Security-Enhanced Linux (SELinux)[۱۴] را معرفی کرد. SELinux یک سیستم امنیتی برای کنترل دسترسی‌ها و استفاده از ACL بود که به‌صورت یک patch به هسته اعمال می‌شد. به‌تدریج با افزایش تعداد patchهایی که برای بهبود سیستم امنیتی هستهٔ Linux معرفی می‌شدند، ضرورت تعریف یک چهارچوب امنیتی برای آن به‌وجود آمد. در این چهارچوب یک‌سری ساختار داده‌ای برای نگهداری خصوصیات امنتی اجزای هسته استفاده می‌شوند و یک‌سری توابع نیز وجود دارند که اعمال صورت‌گرفته بر روی این اجزا را کنترل می‌کنند. هر سیستم امنیتی می‌تواند به‌صورت یک ماجول داخل هسته بار شود و از این ساختار داده‌ای و توابع امنیتی استفاده کرده، مکانیزم خاص خود را پیاده‌سازی کند. یک چنین سیستمی کار برنامه‌نویسان را در توسعهٔ ماجول‌های امنیتی ساده‌تر می‌کند. Linux Security Modules (LSM)[۱۵] به‌عنوان یک چهارچوب امنیتی برای هسته حاصل فعالیت پروژه‌های امنیتی مانند Immunix، SELinux، SGI، Janus و ... می‌باشد. با به‌وجود آمدن LSM بسیاری از پروژه‌های امنیتی از جمله SELinux با اعمال تغییرات لازم در این چهارچوب قرار گرفتند و از LSM به‌عنوان مبنای کار خود استفاده کردند.

۲-۱ مقدمه‌ای بر LSM

LSM یک چهارچوب برای پشتیبانی از ماجول‌های امنیتی می‌باشد. در حال حاضر تاکید LSM روی ماجول‌های کنترل‌کنندهٔ دسترسی می‌باشد، هرچند پیش‌بینی می‌شود در توسعه‌های آینده از نیازهای دیگر امنیتی نیز پشتیبانی کند. LSM به خودی خود هیچ امنیتی به سیستم اضافه نمی‌کند و فقط یک زیربنا برای پشتیبانی از مکانیزم‌های امنیتی دیگر فراهم می‌کند. LSM به‌صورت یک patch برای نسخه‌های 2.4 و 2.5 هسته ارائه می‌شود. با اعمال این patch ساختار داده‌ای و توابع موردنیاز جهت استفادهٔ ماجول‌های امنیتی، همچنین فراخوانی توابع مربوطه در جاهایی که دسترسی‌های خاصی باید تست شود به هسته اضافه می‌شود. به این توابع دراصطلاح

[13] National Security Agency
[14] http://www.nsa.org/selinux
[15] http://lsm.immunix.org

hook گفته می‌شود. کد ماجول‌های امنیتی Dummy، Capability، OW[16]، DTE، LIDS[17] و SELinux نیز همراه LSM وجود دارد که به هسته اضافه می‌شود و هنگام کامپایل کردن آن قابل تنظیم می‌باشند.

ماجول Dummy کوچکترین پیاده‌سازی ممکن از توابع امنیتی LSM می‌باشد. در ماجول Dummy این توابع کار خاصی انجام نمی‌دهند و فقط یک‌سری مقادیر پیشفرض برمی‌گردانند. این ماجول می‌تواند به‌عنوان یک پایه برای پیاده‌سازی ماجول‌های امنیتی مبتنی بر LSM به‌کار رود. این ماجول هنگام اجرا شدن هسته به‌طور اتوماتیک در آن بار می‌شود که بعداً یک ماجول دیگر می‌تواند به‌جای آن قرار بگیرد. ازآنجاکه توابع امنیتی LSM داخل کد هسته فراخوانی می‌شوند، همیشه باید یک ماجول امنیتی برای پاسخ دادن به آنها داخل هسته بار شده باشد حتی اگر هیچ کار خاصی انجام ندهد. ماجول‌های امنیتی قابلیت stack شدن را نیز دارند، بدین معنی که یک ماجول دیگر (فقط یکی) نیز می‌تواند به‌عنوان ماجول امنیتی ثانویه داخل هسته بار شود. البته اجازه و مسئولیت بارگذاری آن با ماجول اولیه می‌باشد.

ساختار داده‌ای security_operations که توسط LSM به هسته اضافه می‌شود شامل یک‌سری اشاره‌گر تابع است، اشاره‌گر به توابعی که داخل کد هسته توسط اجزای مختلف جهت تست کردن دسترسی‌ها و اجازه‌ها فراخوانی می‌شوند. با مقداردهی این اشاره‌گرها هر ماجولی می‌تواند توابع خاص خود را جایگزین توابع LSM نماید. از بین اینها توابع کلی جدا شده و بقیه در ساختارهای دیگری طبقه‌بندی شده‌اند که شامل موارد زیر می‌باشند:

- binprm_security_ops (Security hooks for program execution)
- super_block_security_ops (Security hooks for filesystem operations)
- inode_security_ops (Security hooks for filesystem operations)
- file_security_ops (Security hooks for file operations)
- task_security_ops (Security hooks for task operations)
- socket_security_ops (Security hooks for socket operations)
- skb_security_ops (Lifecycle hooks for network buffers)
- ip_security_ops (IPv4 networking hooks)
- netdev_security_ops (Security hooks for network devices)
- module_security_ops (Security hooks for kernel module operations)
- ipc_security_ops (Security hooks for System V IPC operations)
- msg_msg_security_ops (Security hooks for individual messages held in System V IPC message queues)
- msg_queue_security_ops (Security hooks for System V IPC message queues)

[16] OpenWall
[17] Linux Intrusion Detection System

- shm_security_ops (Security hooks for System V IPC shared memory segments)
- sem_security_ops (Security hooks for System V IPC semaphores)

از جمله مهمترین توابع موجود در security_operations توابع زیر می‌باشند:

- register_security
- unregister_security
- sys_security
- capable

یک اشاره‌گر به‌نام security_ops از جنس security_operations به‌طور سراسری در کد هسته تعریف شده و از این متغیر برای دسترسی به توابع ماجول امنیتی استفاده می‌شود.

توابع امنیتی که به هسته اضافه می‌شوند از قرار زیر می‌باشد:

- register_security و unregister_security : برای اضافه و حذف کردن ماجول امنیتی اولیه به‌کار می‌رود. در ابتدا مقدار dummy_ops (اشاره‌گر متعلق به ماجول Dummy) در security_ops قرار دارد. با اجرای register_security(ops) مقدار ops که اشاره‌گر متعلق به ماجول امنیتی جدید می‌باشد در security_ops ریخته می‌شود و با اجرای unregister_security(ops) مقدار پیشفرض (dummy_ops) در آن قرار می‌گیرد.
- mod_reg_security و mod_unreg_security : این توابع برای اضافه و حذف کردن ماجول امنیتی ثانویه به‌کار می‌رود که در آنها توابع register_security و unregister_security متعلق به ماجول امنیتی اولیه (در ساختار security_operations) فراخوانی می‌شود. به‌این ترتیب stack کردن یک ماجول امنیتی دیگر توسط ماجول اولیه انجام می‌شود.
- sys_security : یک تابع سیستم[۱۸] جدید است که برای تعریف توابع سیستم به‌کار می‌رود و در آن تابع sys_security متعلق به ماجول امنیتی اولیه فراخوانی می‌شود. به این ترتیب ماجول‌ها می‌توانند با استفاده از این تابع، توابع سیستم مورد نیاز خود را تعریف و پیاده‌سازی نمایند.
- capable : این تابع در کد هسته برای کنترل قابلیت‌ها به‌کار می‌رود که در آن تابع capable متعلق به ماجول امنیتی اولیه فراخوانی می‌شود. به این ترتیب ماجول‌ها می‌توانند با پیاده‌سازی این تابع از قابلیت‌ها مطابق نیاز خود استفاده نمایند. درمورد قابلیت‌ها در قسمت ماجول Capability توضیحات کامل داده شده‌است.

[18] System call

جدول شمارهٔ ۱ خصوصیات ماجول‌های امنیتی مختلف را از جهت نحوهٔ بارشدن و قابلیت stackکردن ماجول‌های دیگر به‌طور خلاصه نشان می‌دهد.

جدول ۱ - امکانات ماجول‌های امنیتی مختلف

Operation	CAP	OW	LIDS	DTE	SE
Allow stacking?	-	-	-	+	+
Can be stacked?	+	+	-	-	-

اطلاعات بیشتر درمورد نحوهٔ پیاده‌سازی توابع امنیتی توسط ماجول‌های امنیتی در جدول ضمیمه آمده‌است.

۲-۲ ماجول امنیتی Capability

در علم کامپیوتر قابلیت به اطلاعاتی گفته می‌شود که توسط یک پروسه برای اثبات اینکه قادر به انجام عمل خاصی بر روی یک شیء خاص می باشد به‌کار می‌رود. قابلیت، هر شیء را با مجموعهٔ عملیات مجاز بر روی آن مشخص می‌کند. برای مثال شاخص فایل یک قابلیت است. شاخص فایل توسط تابع سیستم open به‌همراه درخواست اجازه‌های خواندن یا نوشتن ایجاد می‌شود. ساختار داده‌ای مورد نیاز در ابتدا توسط تابع سیستم open ایجاد می‌شود و بعدها از همین ساختار برای چک کردن مجوز عملیات درخواستی بر روی فایل استفاده می‌شود. از شاخص فایل به‌عنوان یک اندیس برای دسترسی به این ساختار داده‌ای استفاده می‌شود. آنچه در اینجا بیشتر مد نظر ما می‌باشد چیزی است که تحت عنوان قابلیت‌های POSIX شناخته شده و Linux از آن استفاده می‌کند. عملکرد این سیستم، تقسیم قدرت root به اجازه‌ها و امتیازات کوچکتر است. در حالت عادی هر پروسه‌ای که نیاز به انجام یک کار سیستمی دارد یا باید با اجازه‌های root اجرا شود و یا دارای اجازهٔ setuid باشد. با استفاده از قابلیت‌های POSIX به هر پروسه فقط اجازه‌هایی که لازم دارد داده می‌شود و نیازی به استفاده از اجازه‌های root نیست.

یک پروسه سه سری نقشهٔ بیتی با نام‌های <قابلیتهای مؤثر>، <قابلیت‌های مُجاز> و <قابلیت‌های موروثی> دارد. هر قابلیت خاص به‌صورت یک بیت در این نقشه‌های بیتی پیاده‌سازی شده است که می‌تواند صفر یا یک باشد. در حالت عادی هنگامی که یک پروسه قصد انجام یک کار سیستمی را دارد uid مؤثر پروسه چک می‌شود که در صورت صفر بودن پروسه اجازهٔ انجام عمل مورد نظر را می‌یابد؛ ولی با استفاده از قابلیت‌های ROSIX بیت متناظر با آن عمل در نقشهٔ قابلیت‌های مؤثر پروسه چک می‌شود که در صورت یک بودن بیت مربوطه این اجازه به پروسه داده می‌شود. برای مثال وقتی یک پروسه قصد عوض کردن ساعت سیستم را

دارد بیت CAP_SYS_TIME (بیت شمارهٔ ۲۵) نقشهٔ قابلیت‌های مؤثر پروسه چک می‌شود که درصورت یک بودن آن، اجازهٔ عوض کردن ساعت سیستم به پروسه داده می‌شود.

تصمیم‌گیری تنها از روی قابلیت‌های مؤثر پروسه انجام می‌گیرد. دو نقشهٔ دیگر (قابلیت‌های موروثی و قابلیت‌های مجاز) در تعیین مقادیر قابلیت‌های مؤثر دخیل‌اند. در نقشهٔ قابلیت‌های مجاز قابلیت‌هایی که پروسه مجاز به استفاده از آنها می‌باشد مشخص شده است. هرکدام از قابلیت‌های مشخص شده در این نقشه می‌تواند در نقشهٔ قابلیت‌های مؤثر پروسه باشد و نبودن هرکدام به این معناست که پروسه قصد استفاده از آن قابلیت خاص را ندارد. یک پروسه فقط می‌تواند قابلیت‌هایی را که در نقشهٔ قابلیت‌های مجاز وجود دارند به قابلیت‌های مؤثر خود اضافه کند. در نقشهٔ قابلیت‌های موروثی قابلیت‌هایی که به پروسه‌های دیگر (فرزندان) منتقل می‌شوند مشخص شده‌اند. اگر یک پروسه، پروسهٔ دیگری را با استفاده از دستور ()exec اجرا کند این قابلیت‌ها به پروسهٔ جدید منتقل می‌شود[۱۹]. یک پروسه فقط می‌تواند قابلیت‌هایی را که در نقشهٔ قابلیت‌های مجاز خود وجود دارد به پروسه‌های دیگر منتقل کند.

فایل‌های اجرایی نیز دارای همان سه نقشهٔ قابلیت‌ها می‌باشند. نقشهٔ قابلیت‌های موروثی، قابلیت‌هایی را مشخص می‌کند که فایل اجرایی می‌تواند به‌هنگام اجرا شدن، از یک پروسهٔ دیگر (پدر) دریافت کند. هنگام منتقل شدن قابلیت‌ها از یک پروسه به پروسهٔ دیگر در ابتدا نقشهٔ قابلیت‌های موروثی پدر و سپس نقشهٔ قابلیت‌های موروثی فرزند بررسی می‌شود و قابلیت‌هایی که در هردوی این نقشه‌ها باشد از پدر به فرزند منتقل می‌شود. نقشهٔ قابلیت‌های مجاز یک فایل اجرایی، قابلیت‌هایی را مشخص می‌کند که پس از اجرا شدن به پروسهٔ حاصل اعطا خواهد گردید. نقشهٔ قابلیت‌های مؤثر می‌تواند تماماً صفر یا تماماً یک باشد، به همین دلیل کل این نقشه با یک بیت نشان داده می‌شود. یک بودن این بیت باعث انتقال قابلیت‌ها از نقشهٔ قابلیت‌های مجاز به نقشهٔ قابلیت‌های مؤثر و صفر بودن آن مانع از این کار می‌شود.

برای محاسبهٔ قابلیت‌های مؤثر پروسه‌ای که از اجرا کردن یک فایل اجرایی حاصل شده چنین عمل می‌شود: در ابتدا نقشهٔ قابلیت‌های موروثی پروسه پدر با نقشهٔ قابلیت‌های موروثی فایل اجرایی and بیتی می‌شود، سپس حاصل با نقشهٔ قابلیت‌های مجاز فایل or بیتی می‌شود. تا اینجای کار قابلیت‌های مجاز پروسهٔ حاصل محاسبه شده است. در نهایت حاصل این عملیات با نقشهٔ قابلیت‌های مؤثر فایل and بیتی می‌شود. حاصل در نقشهٔ قابلیت‌های مؤثر پروسهٔ ایجاد شده کپی می‌شود. فرم ساده شدهٔ این فرایند در فرمول‌های زیر قابل مشاهده است:

---

[۱۴] با اجرای ()fork و ()clone هیچ اتفاقی نمی‌افتد. نقشه قابلیت‌ها دقیقاً بیت بیت به پروسهٔ جدید کپی می‌شود.

Cp = Fp | (Fi & Pi)
Ce = Fe & Cp
(P: Parent Process, C: Child Process, F: File, i: inheritable, p: permitted, e: effective)

هستهٔ Linux از نسخهٔ 2.2 به‌بعد دارای امکان استفاده از قابلیت‌ها می‌باشد. در انتهای فایل وضعیت هر پروسه (/proc/pid/status) می‌توان نقشهٔ قابلیت‌های پروسه‌های درحال اجرای سیستم را مشاهده کرد. قابلیت‌های موجود در آخرین نسخهٔ هسته شامل ۲۷ مورد می‌باشد که در جدول شمارهٔ ۱ لیست شده‌اند.

جدول ۱- لیست قابلیت‌های موجود در هستهٔ Linux

0	CAP_CHOWN	14	CAP_IPC_LOCK
1	CAP_DAC_OVERRIDE	15	CAP_IPC_OWNER
2	CAP_DAC_READ_SEARCH	16	CAP_SYS_MODULE
3	CAP_FOWNER	17	CAP_SYS_RAWIO
4	CAP_FSETID	18	CAP_SYS_CHROOT
5	CAP_KILL	19	CAP_PTRACE
6	CAP_SETGID	20	CAP_SYS_PACCT
7	CAP_SETUID	21	CAP_ADMIN
8	CAP_SETPCAP	22	CAP_SYS_BOOT
9	CAP_LINUX_IMMUTABLE	23	CAP_NICE
10	CAP_NET_BIND_SERVICE	24	CAP_SYS_RESOURCE
11	CAP_NET_BROADCAST	25	CAP_TIME
12	CAP_NET_ADMIN	26	CAP_SYS_TTY-CONFIG
13	CAP_NET_RAW		

مجموعهٔ قابلیت‌هایی که سیستم می‌تواند از آنها استفاده کند یا محدودهٔ قابلیت‌ها[۲۰] داخل کد هستهٔ Linux مشخص می‌شود و معمولاً شامل تمام قابلیت‌ها به استثنای CAP_SETPCAP می‌باشد. مقدار این محدوده را می‌توان تغییر داد ولی با reboot شدن سیستم دوباره مقدار پیشفرض خود را می‌گیرد. برای ایجاد تغییرات دائمی باید در کد هسته تغییرات لازم را اعمال نمود. از روی مقدار عددی که در فایل /proc/sys/kernel/cap-bound قرار دارد می‌توان این محدوده را مشخص کرد. به‌طور پیشفرض پروسهٔ init و تمام پروسه‌هایی که توسط آن اجرا می‌شوند تمام قابلیت‌های موجود در این محدوده را دارا می‌باشند.

[20] Capability bound

ازآنجاکه استفاده از قابلیت‌ها یک امکان نسبتاً جدید می‌باشد از نظر مستندات و نرم‌افزارهای کمکی کمبودهای زیادی در این زمینه وجود دارد. ازجمله برنامه‌های کمکی مفید می‌توان به libcap اشاره کرد[۲۱]. libcap شامل یک کتابخانه برای استفاده از قابلیت‌های هسته در اختیار برنامه‌نویس قرار می‌دهد. با استفاده از این کتابخانه می‌توان برنامه‌هایی نوشت که قادرند قابلیت‌های پروسه‌ها را مدیریت کنند. یک ابزار سودمند دیگر lcap است.[۲۲] lcap قادر است محدودهٔ قابلیت‌های سیستم را تغییر دهد، البته این تغییر فعلاً فقط به‌صورت حذف از میان این محدوده می‌باشد. استفاده از این دستور پس از بوت شدن کامل سیستم بسیار مفید است. برای مثال با استفاده از دستور زیر می‌توان به‌طور کلی مقابل استفاده از قابلیت CAP_SYS_MODULE را گرفت. در این حالت تحت هیچ شرایطی نمی‌توان یک ماجول را با استفاده از دستور insmod داخل هسته بار کرد. بدین ترتیب با غیرفعال کردن قابلیت‌هایی که برای عملکرد عادی سیستم لازم نیستند مقابل بسیاری از رخنه‌های امنیتی گرفته می‌شود.

```
#lcap CAP_SYS_MODULE
```

۲-۳ ماجول امنیتی SELinux

ماجول امنیتی SELinux ساختار امنیتی flask را در محیط Linux پیاده‌سازی کرده است. پیکربندی SELinux در فایل ss_policy/ قرار دارد که شامل سیاست امنیتی مورد استفاده می‌باشد. سیاست امنیتی از کامپایل کردن یک‌سری قوانین امنیتی و بسط ماکروها ایجاد می‌شود. در هر لحظه می‌توان سیاست امنیتی را دوباره کامپایل کرده و داخل سیستم load نمود. SELinux در دو مود development و enforcement می‌تواند کار کند. در حالت اول سیستم مقابل دسترسی‌های غیرمجاز را نمی‌گیرد و فقط آنها را ثبت می‌کند؛ ولی در حالت دوم مقابل دسترسی‌های غیرمجاز گرفته می‌شود. این که SELinux در کدام مود کار کند هنگام کامپایل کردن هسته مشخص می‌شود. اگر هسته در مود development کامپایل شده باشد می‌توان آن را در مود enforcement قرار داد درصورتی‌که برعکس این عمل امکان‌پذیر نیست.

SELinux از پنج جزء اصلی تشکیل شده‌است: سرور امنیتی، AVC، مدیریت PSID فایل‌ها، توابع سیستم جدید، و پیاده‌سازی توابع امنیتی تعریف‌شده در LSM. سرور امنیتی یک واسط برای دسترسی به سیاست امنیتی فراهم می‌کند که باعث می‌شود سایر بخش‌های سیستم مستقل از سیاست امنیتی مورد استفاده باشند. سرور امنیتی از چند تکنولوژی مختلف برای

[21] http://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/
[22] http://www.rocklinux.org/sources/package/base/lcap/

پیاده‌سازی سیاست‌های امنیتی استفاده می‌کند که مهمترین آنها TE، RBAC و MLS[۲۳] می‌باشد؛ البته استفاده از MLS اختیاری است. یک دسترسی خاص با تمام سیاست‌های امنیتی چک می‌شود و درصورتی‌که از دید همهٔ آنها مجاز محسوب شود مجوز مربوطه صادر می‌گردد. از AVC برای نگهداری مجوزهای امنیتی صادرشده و استفاده از آنها در مواقع لزوم استفاده می‌شود که باعث افزایش سرعت عملکرد سیستم می‌شود. واسط‌هایی برای دسترسی سرور امنیتی به‌منظور مدیریت cache در این سیستم تعبیه شده است. توابع امنیتی نیز ازطریق یک واسط دیگر می‌توانند اجازه‌های امنیتی را ازروی AVC چک کنند. برای مدیریت PSIDهای مربوط به فایل‌سیستم از یک جدول تبدیل inode به PSID استفاده می‌شود که در یک فایل نگهداری می‌شود. یک‌سری توابع سیستم جدید توسط SELinux تعریف شده که از طریق sys_security فراخوانی می‌شوند. این توابع اطلاعات موردنیاز برنامه‌های مختلف را در اختیار آنها قرار می‌دهد. البته SELinux یک تابع به‌نام sys_security_selinux را جایگزین sys_security کرده‌است، چون توابع سیستم جدید دسترسی به مقدار ثبات‌های ذخیره‌شده در پشته را لازم دارند و sys_security این اجازه را به آنها نمی‌دهد. توابع امنیتی پیاده‌سازی‌شده اطلاعات امنیتی مربوط به اجزای هسته را مدیریت می‌کند و مجوزهای دسترسی را برروی اعمال هسته اعمال می‌کنند.

۲-۳-۱ ساختار داخلی SELinux

کد LSM شامل دو دایرکتوری lsm-2.4 (یا lsm-2.5) و selinux می‌باشد. دایرکتوری lsm-2.4 حاوی کد هستهٔ patch شده با LSM است که کد ماجول‌های امنیتی نیز در دایرکتوری security داخل آن قرار دارند. فایل‌های پیکربندی flask مربوط به ماجول SELinux در دایرکتوری flask قرار دارند. پس از کامپایل کردن هسته این فایل‌ها در /usr/local/selinux/flask نصب می‌شوند. از این فایل‌ها هنگام کامپایل کردن سیاست امنیتی استفاده می‌شود. فایل security_classes کلاس‌های مختلفی را که برای طبقه‌بندی اشیاء استفاده می‌شود تعریف می‌کند. مجوزهای مربوط به هرکدام از این کلاس‌ها در فایل access_vectors مشخص شده‌است. فایل initial_sids نیز SIDهای اولیهٔ مورداستفادهٔ سیاست امنیتی را تعریف می‌کند.

کدی که در کنار کد هسته و در دایرکتوری selinux قرار دارد جهت کامپایل کردن سیاست امنیتی و برچسب زدن فایل‌ها استفاده می‌شود. فایل‌های پیکربندی سیاست امنیتی در

[23] Multi-Level Security

دایرکتوری policy و فایل‌های موردنیاز برای برچسب زدن فایل‌ها در دایرکتوری setfiles قرار دارند. فایل file_contexts در این دایرکتوری context امنیتی فایل‌ها را براساس نام آنها مشخص می‌کند. برنامهٔ setfiles/setfiles محتوای این فایل را خوانده، از آن جهت برچسب زدن فایل‌ها استفاده می‌کند. دایرکتوری utils شامل یک‌سری برنامهٔ تغییریافته می‌باشد که از توابع سیستم جدید استفاده می‌کنند و می‌توانند با سرور امنیتی SELinux ارتباط برقرار نمایند. برای نمونه دستور ps جدید می‌تواند context امنیتی پروسه‌های درحال اجرای سیستم را نیز به‌دست آورد. برنامهٔ avc_toggle که همراه با این برنامه‌ها نصب می‌شود برای انتقال از مود development به مود enforcement به‌کار می‌رود. برنامهٔ scripts/newrules.pl فایل‌های log سیستم را خوانده و ازروی آنها مجوزهایی را که سیستم به آنها نیاز دارد مشخص می‌کند.

فایل ss_policy که شامل سیاست امنیتی می‌باشد از کامپایل کردن و بسط ماکروهای موجود در فایل‌های مختلف ساخته می‌شود. ماکروهای به‌کاررفته در فایل‌های پیکربندی در دایرکتوری macros تعریف شده‌اند. فایل‌های زیر سیاست امنیتی را تعریف می‌کنند:

- all.te : پیکربندی TE
- rbac : پیکربندی RBAC
- mls : پیکربندی MLS
- users : ارتباط بین کاربران واقعی سیستم و نقش‌های RBAC را تعیین می‌کند.
- constrains : قوانین خاصی به شکل عبارات بولین که بیشتر برای کنترل نقش‌ها به‌کار می‌رود.
- initial_sid_contexts : context امنیتی متناظر با SIDهای اولیه را تعریف می‌کند.
- fs_contexts : context امنیتی یک فایل‌سیستم برچسب نخورده را تعیین می‌کند.
- devfs_contexts : context امنیتی devfs را تعیین می‌کند.
- net_contexts : context امنیتی اشیاء مربوط به شبکه (از جمله آدرس‌ها و پورت‌ها) را تعیین می‌کند.

فایل پیکربندی TE (all.te) بزرگترین فایل پیکربندی SELinux می‌باشد که از ادغام فایل‌های مختلف حاصل می‌شود. قوانین مربوط به اعمال تایپ در دایرکتوری‌های types و domains قرار دارند. تعریف تایپ‌های مورد استفادهٔ سیستم در دایرکتوری types می‌باشد که در فایل‌های device.te، devpts.te، file.te، network.te، nfs.te، procfs.te و security.te تفکیک شده‌است. تعریف domainها و اجازه‌های دسترسی در دایرکتوری domains قرار دارد. قوانین مربوط به کاربران و پروسه‌های آنها در دو فایل user.te و admin.te قرار دارد. دایرکتوری program شامل فایل‌هایی از قبیل apache.te، atd.te، init.te، inetd.te، squid.te، login.te، passwd.te و ...

می‌باشد که در هرکدام از این فایل‌ها قوانین مربوط به یک پروسه یا سرویس خاص تعریف شده‌است. فایل assert.te نیز شامل قوانینی می‌باشد که دست آخر برای چک کردن پیکربندی به‌کار می‌رود.

۲-۳-۲ پیکربندی TE

در مدل TE به هر پروسه یک برچسب domain و به هر شیء یک برچسب تایپ نسبت داده می‌شود. در این مدل با تمام پروسه‌هایی که در یک domain قرار دارند، و نیز تمام اشیائی که دارای یک تایپ مشخص می‌باشند به‌طور یکسان رفتار می‌شود. در حقیقت domainها و تایپ‌ها به‌عنوان کلاس‌های امنیتی سیستم می‌باشند. یک جفت جدول دسترسی مشخص می‌کند که چگونه domainها با هم ارتباط برقرار می‌کنند و نیز چگونه domainها به تایپ‌ها دسترسی پیدا می‌کنند. به هر کاربر اجازه داده می‌شود که از domainهای خاصی استفاده کند. یک برنامه مانند هر شیء دیگر یک برچسب تایپ دارد که مشخص می‌کند این برنامه توسط چه domainهایی می‌تواند اجرا شود. همچنین جداول دسترسی مشخص می‌کند که چه برنامه‌هایی می‌توانند برای ورود به یک domain خاص اجرا شوند.

مدل TE به‌کار رفته در SELinux اندکی با مدل استاندارد آن تفاوت دارد. در این مدل هیچ فرقی بین domain و تایپ وجود ندارد. domain یک تایپ است که می‌تواند به یک پروسه یا اشیاء مرتبط (مانند فایل‌های proc/ یا یک سوکت که توسط پروسه ایجاد می‌شود) نسبت داده شود. یک جدول واحد مجوزهای بین تایپ‌ها را کنترل می‌کند. در فایل‌های پیکربندی فقط به‌منظور خوانایی بیشتر برای پروسه‌ها از لفظ domain به‌جای type استفاده شده؛ هردو به یک صورت تفسیر می‌شوند.

مدل TE به‌کاررفته در SELinux از اطلاعات کلاس‌بندی flask نیز استفاده می‌کند. مجوزهای TE برحسب یک جفت تایپ و نام کلاس صادر می‌شوند. در این حالت سیاست امنیتی می‌تواند با اشیائی که تایپ‌های یکسان دارند ولی متعلق به کلاس‌های مختلف دارند رفتار مختلف داشته باشد؛ برای مثال یک سوکت TCP که توسط یک domain خاص ایجاد شده با یک سوکت raw IP که توسط همان domain ایجاد شده تفاوت دارد. برای انتقال بین domainها نیز از مفهوم کلاس امنیتی استفاده می‌شود. مدل به‌کاررفته در SELinux یک تفاوت دیگر با مدل استاندارد TE دارد که در انتساب کاربران به domainها می‌باشد. در این مدل کاربران مستقیماً به domainها نسبت داده نمی‌شوند، بلکه از مدل RBAC به‌عنوان یک لایه بین کاربران و

domainها استفاده می‌شود. در این مدل از TE فقط نقش‌های RBAC به domainها منتسب می‌شوند، ارتباط بین نقش‌ها و کاربران را نیز RBAC تعریف می‌کند.

قوانین TE شامل تعریف تایپ، قوانین انتقال تایپ، قوانین تغییر تایپ، قوانین دسترسی و قوانین assertion می‌باشد.

- تعریف تایپ

در پیکربندی TE تمام تایپ‌های مورد استفاده باید تعریف شوند. ازآنجاکه کامپایلر در دو فاز قوانین پیکربندی را مرور می‌کند قبل از تعریف یک تایپ می‌توان به آن ارجاع داد. هر تعریف تایپ یک نام اصلی برای تایپ، یک‌سری نام مستعار اختیاری و یک‌سری خصوصیت برای آن تایپ مشخص می‌کند:

```
type_decl        ➔  TYPE identifier opt_alias_def opt_attr_list ';'
opt_alias_def    ➔  ALIAS aliases | empty
aliases          ➔  identifier | '{' identifier_list '}'
identifier_list  ➔  identifier | identifier_list identifier
opt_attr_list    ➔  ',' attr_list | empty
attr_list        ➔  identifier | attr_list ',' identifier
```

برای روشن‌تر شدن موضوع نمونه‌هایی از تعریف تایپ مربوط به برنامهٔ SSH در زیر آمده‌است:

```
type sshd_t, domain, privuser, privrole, privlog, privowner;
type sshd_exec_t, file_type, exec_type, sysadmfile;
type sshd_tmp_t, file_type, sysadmfile, tmpfile;
type sshd_var_run_t, file_type, sysadmfile, pidfile;
```

sshd_t نام domain متعلق به پروسهٔ SSH می‌باشد. sshd_exec_t تایپ برنامهٔ اجرایی سرویس SSH می‌باشد. sshd_tmp_t و sshd_var_run_t تایپ مربوط به فایل‌های موقت تولیدشده توسط برنامهٔ سرور می‌باشد.

- قوانین انتقال تایپ

قوانین انتقال، تایپ جدید برای یک شیء و یا domain جدید برای یک پروسه تعیین می‌کند. در هر حالت تایپ جدید براساس یک جفت تایپ و یک کلاس امنیتی تعیین می‌شود. برای یک پروسه تایپ اول یا تایپ مبدأ، domain فعلی می‌باشد و تایپ دوم یا تایپ مقصد، تایپ برنامهٔ اجرائی می‌باشد. برای یک شیء تایپ مبدأ، domain پروسهٔ ایجادکننده می‌باشد و تایپ مقصد، تایپ یک شیء مربوط می‌باشد (مثلاً دایرکتوری پدر درمورد یک فایل). فرمت قوانین انتقال تایپ چنین است:

type_transition_rule	➔	TYPE_TRANSITION source_types target_types ':' classes new_type ';'
source_types	➔	set
target_types	➔	set
classes	➔	set
new_type	➔	identifier
set	➔	'*' | identifier | '{' identifier_list '}' | '~' identifier | '~' '{' identifier_list '}'

در این فرمت به‌جای نام تایپ می‌توان یک صفت خاص را به‌کار برد و این کار باعث می‌شود تمام تایپ‌هایی که دارای آن صفت خاص می‌باشند مشمول قانون شوند. به‌کار بردن * باعث انتخاب تمام تایپ‌ها می‌شود و استفاده از علامت ~ باعث انتخاب تمام تایپ‌ها به‌جز تایپ‌های ذکرشده می‌گردد. اگر برای یک جفت تایپ و یک کلاس امنیتی بیش از یک قانون انتقال تعریف شده باشد آخرین قانون اعمال می‌شود. چند نمونه از قوانین انتقال تایپ در زیر آمده‌است:

```
type_transition initrc_t sshd_exec_t:process sshd_t;
type_transition sshd_t tmp_t:{ dir file lnk_file sock_file fifo_file } sshd_tmp_t;
type_transition sshd_t shell_exec_t:process user_t;
```

initrc_t نام domain پروسهٔ init هنگام اجرای برنامه‌های /etc/rc.d/ است. قانون اول تعیین می‌کند که این پروسه پس از اجرای برنامهٔ sshd که تایپ sshd_exex_t دارد وارد domain با نام sshd_t شود. tmp_t تایپ دایرکتوری /tmp/ می‌باشد. قانون دوم تعیین می‌کند که تمام فایل‌ها و زیردایرکتوری‌هایی که در این دایرکتوری توسط پروسهٔ sshd تولید می‌شوند باید دارای تایپ sshd_tmp_t باشند. shell_exec_t تایپ برنامهٔ اجرایی shell (/bin/sh، /bin/bash، /bin/tcsh، ...) می‌باشد. قانون سوم مشخص می‌کند که برنامهٔ sshd پس از اجرای برنامهٔ shell باید وارد domain با نام user_t شود.

- قوانین دسترسی

قوانین دسترسی یک‌سری مجوزها را براساس یک جفت تایپ و یک کلاس امنیتی مشخص می‌کنند. این قوانین جدول دسترسی TE را ایجاد می‌کنند. فرمت قوانین دسترسی چنین است:

te_av_rule	➔	av_kind source_types target_types ':' classes permissions ';'
av_kind	➔	ALLOW | AUDITALLOW | AUDITDENY
source_types	➔	set
target_types	➔	set
classes	➔	set
permissions	➔	set
set	➔	'*' | identifier | '{' identifier_list '}' | '~' identifier | '~' '{' identifier_list '}'

اگر برای یک جفت تایپ و یک کلاس امنیتی بیش از یک قانون دسترسی تعریف شده باشد اجتماع دسترسی‌های تعریف‌شده اعمال می‌شود. چند نمونه از قوانین دسترسی در زیر آمده‌است:

```
allow sshd_t sshd_exec_t:file { read execute };
allow sshd_t sshd_tmp_t:file { create read write getattr setattr link unlink rename };
allow sshd_t user_t:process transition;
```

قانون اول مشخص می‌کند که پروسهٔ sshd می‌تواند فایل‌های با تایپ sshd_exec_t را خوانده و اجرا کند. قانون دوم مشخص می‌کند که پروسهٔ sshd اجازهٔ ایجاد و دسترسی به فایل‌های با تایپ sshd_tmp_t را دارد. قانون سوم مشخص می‌کند که پروسهٔ sshd می‌تواند وارد یک domain با نام user_t شود.

- قوانین assertion

این قوانین برای کنترل دسترسی‌ها و مجوزهای تعریف‌شده به‌کار می‌روند. این قوانین مجوزهایی را که به‌ازای یک جفت تایپ و یک کلاس امنیتی نباید در قوانین دسترسی باشد مشخص می‌کنند. اگر هرکدام از مجوزهای مشخص‌شده در قوانین دسترسی تعریف شده باشند دراین‌صورت سیاست امنیتی کامپایل نمی‌شود. فرمت این قوانین مشابه قوانین دسترسی می‌باشد. چند نمونه از قوانین assertion در زیر آمده‌است:

```
neverallow ~{ kmod_t insmod_t rmmod_t ifconfig_t } self:capability sys_module;
neverallow local_login_t ~login_exec_t:file entrypoint;
```

قانون اول مشخص می‌کند که فقط domainهای خاصی حق استفاده از قابلیت CAP_SYS_MODULE را دارند. قانون دوم مشخص می‌کند که domain با نام local_login_tفقط ازطریق اجرای برنامهٔ با تایپ login_exec_t قابل دستیابی است.

۲-۳-۳ پیکربندی RBAC

در مدل استاندارد RBAC به کاربران اجازهٔ استفاده از نقش‌های خاصی داده می‌شود و به هر نقش دسترسی‌های خاصی اعطا می‌گردد. در مدل RBAC به‌کاررفته در SELinux دسترسی‌ها مستقیماً به نقش‌ها داده نمی‌شود بلکه نقش‌ها خود با domainهای مدل TE در ارتباطند که مجوزها و دسترسی‌های مجاز را مشخص می‌کنند. در context امنیتی هر پروسه یک فیلد مربوط به نقش‌های RBAC وجود دارد. مقدار این فیلد برای سایر اشیاء مقدار کلی object_r می‌باشد که از آن استفاده نمی‌شود. در RBAC ارتباط سلسله مراتبی بین نقش‌ها وجود دارد.

علاوه بر این انتقال نقش‌ها از طریق ترکیبی از قوانین RBAC و TE صورت می‌گیرد؛ فقط domainهای خاصی اجازهٔ این کار را دارند.

قوانین RBAC شامل تعریف نقش و کنترل دسترسی نقش‌ها می‌باشد.

- قوانین تعریف نقش

RBAC یک نقش را از طریق قوانین تعریف نقش به یک‌سری تایپ نسبت می‌دهد. فرمت قوانین تعریف نقش چنین است:

```
role_decl   ➔  ROLE identifier TYPES types ';'
types       ➔  set
set         ➔  '*' | identifier | '{' identifier_list '}' | '~' identifier |
               '~' '{' identifier_list '}'
```

چند نمونه از قوانین تعریف نقش در زیر آمده‌است:

```
role system_r types { kernel_t initrc_t getty_t klogd_t };
role user_r types { user_t user_netscape_t };
role sysadm_r types { sysadm_t run_init_t };
```

- قوانین کنترل نقش‌ها

این نوع قوانین انتقال بین نقش‌ها را کنترل می‌کنند. فرمت این قوانین چنین است:

```
role_allow_rule  ➔  ALLOW current_roles new_roles ';'
current_roles    ➔  set
new_roles        ➔  set
set              ➔  '*' | identifier | '{' identifier_list '}' | '~' identifier |
                    '~' '{' identifier_list '}'
```

چند نمونه از قوانین کنترل نقش‌ها در زیر آمده‌است:

```
allow system_r { user_r sysadm_r };
allow user_r sysadm_r;
allow sysadm_r system_r;
```

۴-۳-۲ ماکروها

برای سادگی پیکربندی یک‌سری ماکرو تعریف شده‌است که از آنها می‌توان در نوشتن قوانین استفاده کرد. یک دسته این ماکروها برای مشخص کردن یک مجموعه از کلاس‌ها به‌کار می‌روند، مانند ماکروهای زیر:

```
dir_file_class_set
file_class_set
notdevfile_class_set
devfile_class_set
socket_class_set
dgram_socket_class_set
stream_socket_class_set
unpriv_socket_class_set
```

یک دسته از ماکروها برای مشخص کردن یک مجموعه از کلاس‌ها به‌کار می‌روند، مانند ماکروهای زیر:

```
file
   stat_file_perms
   x_file_perms
   r_file_perms
   rx_file_perms
   rw_file_perms
   ra_file_perms
   link_file_perms
   create_file_perms
   r_dir_perms
   rw_dir_perms
   ra_dir_perms
   create_dir_perms
   mount_fs_perms
socket
   rw_socket_perms
   create_socket_perms
   rw_stream_socket_perms
   create_stream_socket_perms
IPC
   r_sem_perms
   rw_sem_perms
   r_msgq_perms
   rw_msgq_perms
   r_shm_perms
   rw_shm_perms
```

یک دسته از ماکروها برای تعریف قوانین دسترسی به‌کار می‌روند، با استفاده از این ماکروها می‌توان بسیاری از دسترسی‌های متداول را در قالب یک سطر خلاصه کرد. یک نمونهٔ خوب ماکروی domain_auto_trans می‌باشد. برای این‌که domain با نام sshd_t بتواند با اجرای یک برنامه با تایپ shell_exec_t وارد domain با نام user_t شود می‌توان از سطر زیر استفاده کرد:

```
domain_auto_trans(sshd_t, shell_exec_t, user_t)
```

نمونه‌های دیگر از این نوع ماکروها از قرار زیر می‌باشد:

general_domain_access
general_proc_read_access
general_file_read_access
general_file_write_access
uses_shlib
can_network
every_domain
domain_trans and domain_auto_trans
file_type_trans and file_type_auto_trans
can_exec
can_exec_any
can_unix_connect
can_unix_send
can_tcp_connect
can_udp_send
can_sysctl
can_create_pty
can_create_other_pty

۴-۲ ماجول‌های دیگر

ترکیب ماجول Capability و SELinux می‌تواند امنیت مناسبی را برای هستهٔ Linux فراهم کند، ولی ماجول‌های دیگری نیز وجود دارند که از آنها نیز می‌توان استفاده کرد. از جملهٔ این ماجول‌ها می‌توان به Openwall، DTE و LIDS[۲۴] اشاره کرد.

- ماجول امنیتی Openwall

این ماجول مجموعه‌ای از patchهای امنیتی می‌باشد که مهمترین آنها برای مقابله با حملات سرریز بافر کاربرد دارد. این نوع حملات معمولاً به دو روش صورت می‌گیرند. در روش اول آدرس بازگشت یک تابع روی پشته طوری عوض می‌شود که پس از خروج از تابع یک کد دلخواه که معمولاً در پشته قرار دارد اجرا شود. با غیرقابل اجرا کردن کد روی پشته جلوی این نوع حمله گرفته می‌شود. در روش دیگر آدرس بازگشت تابع طوری عوض می‌شود که یکی از توابع libc (معمولاً system) اجرا شود. این توابع از طریق آدرس‌های خاص حافظه قابل دسترسی هستند. با صفر کردن آدرس‌های نسبت داده‌شده به توابع libc جلوی این نوع حمله نیز گرفته می‌شود. از

[24] http://www.lids.org

امکانات دیگر Openwall می‌توان به کنترل لینک‌ها و pipeها در دایرکتوری tmp/، کنترل فایل‌های proc/، کنترل شاخص‌های فایل و کنترل حافظهٔ مشترک اشاره کرد.

- ماجول امنیتی DTE

این ماجول تکنولوژی TE (DTE) را پیاده‌سازی کرده‌است.

- ماجول امنیتی LIDS

ماجول امنیتی LIDS از طرق مختلف جلوی حملات و دسترسی‌های غیرمجاز را می‌گیرد. اقدامات امنیتی این ماجول در سه دستهٔ مختلف قرار دارند: محافظت از سیستم، تشخیص نفوذ و عکس‌العمل درمقابل حملات.

در LIDS بسیاری از اجزای سیستم غیرقابل دسترسی می‌باشند. دسترسی مستقیم به حافظه (/dev/mem، /dev/kcore، ...)، دسترسی مستقیم به دیسک (/dev/hdxx، /dev/sdxx، ...)، دسترسی مستقیم به ورودی/خروجی (/dev/port، توابع سیستم iopl و ioperm، ...) و بسیاری از دسترسی‌های مشابه دیگر امکان‌پذیر نمی‌باشند. بسیاری از اجزای سیستم مانند برنامه‌های اجرایی، فایل‌های log و فایل‌های موردنیاز برای بوت شدن سیستم طوری محافظت شده‌اند که قابل تغییر نمی‌باشند، حتی توسط کاربر root. فایل‌های پیکربندی، جداول مسیریابی و فایل‌سیستم‌های mount شده همگی در این سیستم محافظت شده‌اند. ماجول‌های هسته در ابتدا هنگام بوت شدن سیستم توسط init بارگذاری می‌شوند و بعداً امکان حذف یا اضافه کردن ماجول‌ها وجود ندارد. درحالت کلی هر مجوز و دسترسی که سیستم به آن نیاز ندارد گرفته می‌شود. برای مدیریت سیستم و اعمال تغییرات می‌توان با وارد کردن یک password خاص سیستم امنیتی LIDS را غیرفعال کرد و پس از اعمال تغییرات دوباره آن را به حالت اول بازگرداند.

LIDS می‌تواند از قابلیت‌های هسته نیز برای کنترل عملکرد سیستم استفاده کند. با کنترل محدودهٔ قابلیت‌ها می‌توان رفتار سیستم را محدودتر کرد. برای مثال سیستمی که شبکهٔ آن به‌خوبی پیکربندی و تنظیم شده نیازی ندارد که از قابلیت CAP_NET_ADMIN استفاده کند. به‌این ترتیب امکانات موجود در قابلیت‌های هسته را می‌توان با امکانات LIDS ترکیب کرد.

LIDS علاوه بر محافظت از سیستم دربرابر حملات، از مکانیزم‌هایی برای ثبت کردن دسترسی‌های غیرمجاز و موارد مشکوک استفاده می‌کند. حتی در صورت بروز موارد مشکوک این

امکان وجود دارد که هشدارهای امنیتی خاصی تولید شود. این هشدارها می‌توانند به‌صورت نامهٔ الکترونیکی برای یک آدرس خاص فرستاده شوند.

ضمیمه

جدول شمارهٔ ۲ تمام فیلدهای ساختار داده‌ای security_operations را لیست کرده، همچنین نحوهٔ پیاده‌سازی آنها توسط ماجول‌های امنیتی مختلف را نشان می‌دهد. مقدار خانه‌های جدول بیانگر نحوهٔ پیاده‌سازی یک تابع توسط یک ماجول خاص می‌باشد که می‌تواند مقادیر زیر را داشته باشد:

- خالی : تابع مورد نظر در ماجول پیاده‌سازی نشده‌است.
- I : عملکرد تابع مورد نظر در ماجول پیاده‌سازی شده‌است.
- C : عملکرد تابع از طریق قابلیت‌ها (تابع capable) کنترل می‌شود.
- S : تابع متناظر در ماجول ثانویه (stack شده) را فراخوانی می‌کند.

جدول ۲ - نحوهٔ پیاده‌سازی توابع امنیتی

Function	CAP	OW	LIDS	DTE	SE
binprm_security_ops					
alloc_security		I		I	I
free_security		I		I	I
compute_creds	I	I		S	IS
set_security	I	I		I	IS
check_security			I		
super_block_security_ops					
alloc_security				I	I
free_security				I	I
statfs					I
mount				I	I
check_sb				I	I
umount				I	I
umount_close					I
umount_busy					I
post_remount			I		I
post_mountroot				I	I
post_addmount				I	I
inode_security_ops					
alloc_security				I	I
free_security				I	I
create					I
post_create				I	I
link		I			I
post_link					
unlink			I		I
symlink			I		I
post_symlink				I	I
mkdir			I		I
post_mkdir				I	I
rmdir			I		I
mknod			I		I
post_mknod				I	I
rename			I		I
post_rename					

readlink			I		I
follow_link		I	I	S	I
permission			I	I	I
revalidate					
setattr			I		I
stat					I
post_lookup				I	I
delete					I
file_security_ops					
permission			I		I
alloc_security					I
free_security					I
llseek					I
ioctl					I
mmap					I
mprotect					I
lock					I
fcntl					I
set_fowner					I
send_sigiotask					I
receive					I
task_security_ops					
create					I
alloc_security			I	I	I
free_security			I	I	I
setuid					
post_setuid	I			S	S
setgid					
setpgid					I
getpgid					I
getsid					I
setgroups					
setnice					I
setrlimit					
setscheduler					I
getscheduler					I
kill			I	I	I
wait					I
prctl					
kmod_set_label	I			S	IS
socket_security_ops					
create					I
post_create					I
bind					I
connect					I
listen					I
accept					I
sendmsg					I
recvmsg					I
getsockname					I
getpeername					I
getsockopt					I
setsockopt					I
shutdown					I
sock_rcv_skb					I
unix_stream_connect					I
unix_may_send					I
skb_security_ops					
alloc_security					I
clone					I

copy					I
set_owner_w					I
free_security					I
ip_security_ops					
preroute_first					
preroute_last					I
input_first					I
input_last					I
forward_first					
forward_last					
output_first					I
output_last					
postroute_first					
postroute_last					I
fragment					I
defragment					I
encapsulate					
decapsulate					
decode_options					I
netdev_security_ops					
unregister					I
module_security_ops					
create_module	C		C	C	C
init_module	C		C	C	C
delete_module	C		C	C	C
ipc_security_ops					
permission					I
getinfo					I
msg_msg_security_ops					
alloc_security					I
free_security					I
msg_queue_security_ops					
alloc_security					I
free_security					I
associate					I
msgctl					I
msgsnd					I
msgrcv					I
shm_security_ops					
alloc_security					I
free_security					I
associate					I
shmctl					I
shmat					I
sem_security_ops					
alloc_security					I
free_security					I
associate					I
semctl					I
semop					I
security_operations					
sethostname	C		C	C	C
setdomainname	C		C	C	C
reboot	C		C	C	C
ioperm	C		C	C	C
iopl	C		C	C	C
ptrace	I			S	IS
capget	I			S	IS
capset_check	I			S	IS
capset_set	I			S	IS

acct	C		C	C	C
sysctl					I
capable	I		I	S	IS
sys_security				I	I
swapon					I
swapoff	C		C	C	C
nfsservctl					I
quotactl					I
quota_on					I
bdflush					I
syslog					I
netlink_send					I
netlink_recv					I
register_security					I
unregister_security					I