Pokročilé bezpečnostné mechanizmy OS Linux

Jaroslav Janáček

UNIVERZITA KOMENSKÉHO V BRATISLAVE
2022

Pokročilé bezpečnostné mechanizmy OS Linux

Autor: Jaroslav Janáček
Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra informatiky

Recenzenti: doc. Ing. Pavel Segeč, PhD.
doc. Ing. Anton Baláž, PhD.

Vydavateľ: Univerzita Komenského v Bratislave
Bratislava 2022
1. vydanie

Materiál je výstupom Rozvojového projektu Univerzity Komenského a Ministerstva školstva, vedy, výskumu a športu SR č. 002 UK-2-1/2018 – *„Vzdelávanie pre informačnú spoločnosť“* v oblasti Podpora vysokých škôl pri plnení záväzkov prijatých v rámci Národnej koalície pre digitálne zručnosti a povolania SR.

ISBN 978-80-223-5406-6

Obsah

Úvod

Nasadzovanie OS Linux do produkčného prostredia vyžaduje, aby tvorcovia aplikačného programového vybavenia, systémoví integrátori, ako aj systémoví administrátori venovali dostatočnú pozornosť otázkam bezpečnosti. S rastúcou komplexnosťou informačných systémov rastie aj priestor pre výskyt chýb, či už chýb priamo v návrhu riešení, alebo chýb pri implementácii. Takéto chyby často vytvárajú zraniteľnosti informačného systému. S rastúcim množstvom zaujímavých údajov spracúvaných v informačných systémoch rastie motivácia rôznych útočníkov na prípravu a (pokusy o) realizáciu rôznych útokov využívajúcich takého zraniteľnosti. Pri návrhu, implementácii a prevádzke informačných systémov je preto žiadúce efektívne využívať existujúce bezpečnostné mechanizmy poskytované operačným systémom tak, aby sa redukovali bezpečnostné riziká plynúce z (predpokladanej) existencie zraniteľností v informačnom systéme.

Základný bezpečnostný model OS Linux vychádzajúci z POSIX štandardov je pomerne dobre známy. Menej známe sú však jeho viaceré rozšírenia, ktoré pri správnom použití umožňujú značne redukovať potenciálny dopad zneužitia rôznych zraniteľností v aplikačných a systémových programoch. Cieľom prvej časti tejto knihy je poskytnúť čitateľovi ucelený pohľad na prostriedky riadenia prístupu v OS Linux – od základného modelu voliteľného riadenia prístupu, cez jeho rozšírenia v podobe ACL, cez subsystém Capabilities, až po bezpečnostné moduly SELinux a AppArmor. Predpokladá sa, že čitateľ má aspoň základné znalosti z oblasti operačných systémov a administrácie operačných systémov UNIX-ového typu.

1. Základy riadenia prístupu v OS Linux

Základom riadenia prístupu je rozhodovanie, či *subjekt* (aktívna entita) môže alebo nemôže vykonať požadovanú *operáciu* s *objektom* (pasívnou entitou).

Subjektami riadenia prístupu v OS Linux sú *procesy*, ktoré vždy operácie vykonávajú v mene nejakého *používateľa*. Používatelia reprezentujú skutočných (ľudských) používateľov alebo virtuálnych používateľov (napr. predstavujúcich nejaký subsystém informačného systému). Používatelia môžu byť členmi *skupín* používateľov. V základnom modeli voliteľného riadenia prístupu v OS Linux sú prístupové práva prideľované používateľom a skupinám. Proces, ktorý vykonáva operácie v mene nejakého používateľa, využíva prístupové práva pridelené tomuto používateľovi, resp. skupinám, ktorých je tento používateľ členom. Preto za subjekty riadenia prístupu môžeme považovať aj používateľov a skupiny používateľov.

Objektami riadenia prístupu v OS Linux sú predovšetkým objekty súborového systému (súbory, adresáre, špeciálne súbory reprezentujúce zariadenia, pomenované rúry (pipe, FIFO), pomenované socket-y), ale aj zdieľaná pamäť a iné prostriedky medziprocesovej komunikácie a synchronizácie, procesy (ako cieľ operácie, napr. poslanie signálu alebo použitie volania ptrace) a globálne parametre systému (napr. čas, sieťové parametre, ...). V rámci tejto kapitoly sa zameriame predovšetkým na objekty súborového systému. Operácie so zdieľanou pamäťou a inými prostriedkami medziprocesovej komunikácie sú kontrolované veľmi podobne. Operácie s procesmi sú typicky kontrolované v zmysle, že proces môže vykonať operáciu s iným procesom, ak tieto patria rovnakému používateľovi, alebo ak aktívny proces je privilegovaný (t.j., bez využitia pokročilých prostriedkov, ktorým sa budeme venovať v ďalších kapitolách, je vykonávaný v mene používateľa root). Prístup ku globálnym parametrom systému je typicky kontrolovaný tak, že meniť (a v niektorých prípadoch aj čítať) tieto parametre môže len privilegovaný proces.

1.1. Základné prístupové práva

Každý proces v OS Linux má priradených niekoľko atribútov identifikujúcich používateľa (UID – user ID):

- reálne (real) UID – identifikuje používateľa, ktorý daný proces spustil,
- efektívne (effective) UID – identifikuje používateľa, v mene ktorého proces aktuálne vykonáva operácie,
- uložené (saved) UID – v špeciálnych prípadoch slúži na uloženie UID používateľa, ktorého identitu môže proces na seba prevziať,

- súborové (filesystem) UID (FSUID) – identifikuje používateľa, v mene ktorého proces aktuálne vykonáva operácie s objektami súborového systému; typicky je rovnaké ako efektívne UID.

Obyčajne sú všetky 4 UID rovnaké a identifikujú používateľa, ktorý vytvoril proces – spustil program. Prípadom, keď tomu tak nie je, sa budeme venovať neskôr.

Každý proces má priradených niekoľko atribútov identifikujúcich skupiny používateľov (GID – group ID):

- reálne (real) GID – identifikuje primárnu skupinu používateľa, ktorý proces spustil,
- efektívne (effective) GID – identifikuje primárnu skupinu používateľa, v mene ktorého proces aktuálne vykonáva operácie,
- uložené (saved) GID – v špeciálnych prípadoch slúži na uloženie GID skupiny, ktorej identitu môže proces na seba prevziať,
- súborové (filesystem) GID (FSGID) – identifikuje primárnu skupinu používateľa, v mene ktorého proces aktuálne vykonáva operácie s objektami súborového systému; typicky je rovnaké ako efektívne GID,
- zoznam doplnkových GID (supplementary GIDs) – identifikuje ďalšie skupiny, ktorých členom je používateľ, v mene ktorého proces vykonáva operácie, a teda ktorých práva môže proces využívať.

Obyčajne sú reálne GID, efektívne GID, FSGID aj uložené GID rovnaké a identifikujú primárnu skupinu používateľa, ktorý proces vytvoril. Prípadom, keď tomu tak nie je, sa budeme tiež venovať neskôr.

Každý objekt súborového systému má priradených niekoľko atribútov, ktoré sa používajú na riadenie prístupu:

- vlastník (owner) – UID používateľa, ktorý je vlastníkom objektu – ktorý objekt vytvoril,
- skupina (group) – GID jednej skupiny, ktorej členovia majú mať iné prístupové práva ako ostatní,
- prístupové práva (permission bits) – bitový vektor určujúci oprávnenia jednotlivých subjektov na jednotlivé operácie s týmto objektom:
  - prístupové práva pre vlastníka,

- prístupové práva pre skupinu,
- prístupové práva pre ostatných,
- 3 špeciálne bity set-UID, set-GID, sticky bit.

Prístupové práva pre vlastníka, skupinu a ostatných pozostávajú každé z troch bitov, ktoré určujú práva na

- čítanie (read, r),
- zápis (write, w),
- spustenie programu / použitie adresára v ceste k iného objektu (execute / use, x).

Prístupové právo na čítanie oprávňuje proces čítať obsah súboru. V prípade adresára oprávňuje na čítanie obsahu adresára, t.j. názvov objektov, ktoré sa v adresári nachádzajú a ich identifikačného čísla.

Prístupové právo na zápis oprávňuje proces zapisovať do súboru – meniť jeho obsah. V prípade adresára oprávňuje na zmenu jeho obsahu, t.j. vytváranie nových objektov v adresári, odstraňovanie existujúcich objektov z adresára a premenovanie existujúcich objektov v adresári.

Prístupové právo na spustenie programu oprávňuje proces na spustenie programu uloženého v súbore. V prípade priamo spustiteľných programov (tzv. binárok, binaries) nie je na spustenie programu potrebné právo na čítanie. To umožňuje povoliť používateľom spúšťať programy, ktoré nemôžu prečítať, a teda ani kopírovať, analyzovať, a pod. V prípade skriptov však právo na spustenie nestačí, tu je potrebné aj právo na čítanie (dôvodom je, že v prípade skriptu sa jeho spustenie realizuje spustením príslušného interpretera skriptovacieho jazyka, ktorý následne potrebuje prečítať obsah skriptu, aby ho mohol interpretovať a vykonať).

Prístupové právo na použitie adresára v ceste k inému objektu je potrebné na vykonanie akejkoľvek operácie s objektom, ktorý identifikujeme pomocou cesty, ktorá obsahuje tento adresár. Inými slovami, ak má proces vykonať akúkoľvek operáciu s nejakým objektom, musí mať právo na použitie všetkých adresárov na ceste od koreňa adresárového stromu až po adresár, ktorý obsahuje daný objekt.

Je dobré si uvedomiť niektoré dôsledky vyššie uvedeného významu prístupových práv. Na vymazanie alebo premenovanie existujúceho objektu proces nepotrebuje mať žiadne právo k tomuto objektu, ale potrebuje právo na zápis do adresára, kde sa objekt nachádza. Zároveň potrebuje aj právo na použitie všetkých adresárov v ceste od koreňa. Ak teda máme právo na zápis

do príslušného adresára aj právo na použitie všetkých potrebných adresárov, môžeme vymazať alebo premenovať aj súbor, ktorý nevieme ani čítať, ani modifikovať. Na vytvorenie nového súboru je tiež potrebné právo na zápis do príslušného adresára a právo na použitie adresárov v ceste. Treba si uvedomiť, že operácie vyzamania súboru a vytvorenia súboru spolu umožňujú nahradenie existujúceho súboru novým, čo v mnohých prípadoch môže mať rovnaké dôsledky ako prípadná modifikácia existujúceho súboru. Rozdiel je v tom, že zmena súboru jeho nahradením novým súborom nebude mať vplyv na procesy, ktoré pôvodný súbor už majú otvorený – kým ho nezatvoria, budú pracovať s pôvodným súborom.

Ďalšie dôsledky súvisia s organizáciou informácií o objektoch v súborových systémoch UNIX-ového typu. Adresáre obsahujú len mená objektov a ich identifikačné čísla, tzv. *inode number*. Ďalšie informácie o objekte (ako napr. typ, veľkosť, vlastník, skupina, práva, čas modifikácie, prístupu, …) sú uložené v inej dátovej štruktúre (inode) adresovanej práve týmto identifikačným číslom. Na zistenie informácií z tejto dátovej štruktúry o objekte je potrebné mať právo na použitie adresárov na ceste k tomuto objektu, ale nie je potrebné žiadne ďalšie právo na tento objekt. Z toho vyplýva zaujímavý dôsledok, že ak máme právo na čítanie adresára, ale nemáme právo na použitie tohto adresára v ceste, tak môžeme získať zoznam objektov v adresári a ich identifikačné čísla, ale nemôžeme získať ďalšie informácie o týchto objektoch. Naopak, ak máme právo na použitie adresára v ceste, ale nemáme právo na čítanie adresára, nemôžeme si nechať vypísať zoznam objektov v adresári, ale ak ich mená poznáme, môžeme k nim pristupovať – napr. zistiť o nich informácie z inode, alebo, ak máme príslušné právo, čítať / modifikovať ich obsah, a pod.

Jadro OS Linux vyhodnocuje prístupové práva k objektu súborového systému prakticky pri všetkých operáciách okrem operácií s už otvoreným objektom (čiže napr. práva sa kontrolujú pri otvorení súboru, ale už nie pri následnom čítaní alebo zápise doň). Základný algoritmus vyhodnotenia prístupových práv je nasledovný:

- Ak je UID vlastníka objektu rovnaké ako FSUID procesu, použijú sa práva pre vlastníka,
- inak, ak je GID skupiny objektu rovnaké ako FSGID procesu alebo ako GID niektorej skupiny zo zoznamu doplnkových skupín procesu, použijú sa práva pre skupinu,
- inak sa použijú práva pre ostatných.
- Týmto postupom vybratá trojica bitov reprezentujúca prístupové práva sa porovná s požadovanými právami podľa požadovanej operácie. Ak obsahuje všetky práva, ktoré sú pre operáciu potrebné, operácia sa povolí, inak sa zamietne.

Z vyššie uvedeného algoritmu vyplýva, že vždy sa použije len jedna trojica bitov reprezentujúcich prístupové práva, čiže nekombinujú sa napr. práva pre vlastníka s právami pre skupinu, ani ak vlastník je členom príslušnej skupiny. Taktiež práva pre ostatných sa aplikujú len na také procesy, ktoré nespĺňajú podmienky na aplikáciu práv pre vlastníka ani práv pre skupinu.

Špeciálnou výnimkou sú privilegované procesy – procesy s FSUID = 0, čiže procesy, ktoré vykonávajú súborové operácie v mene administrátora – používateľa root. V takom prípade platí, že sa nekontrolujú práva na čítanie, zápis a použitie adresára v ceste – automaticky sa pokladajú za povolené. Spustenie programu je pre takéto procesy povolené vtedy, ak je povolené aspoň pre niekoho (teda ak vlastník, skupina alebo ostatní majú právo na spustenie programu). Neskôr uvidíme, ako je táto výnimka modifikovaná inými bezpečnostnými mechanizmami. Dôvodom, prečo ani root nemôže spustiť ľubovoľný súbor je to, aby sa omylom nepokúsil spustiť súbor, ktorý nie je spúšťateľný.

1.1.1. Zobrazenie prístupových práv

Atribúty objektu súborového systému môžeme zistiť napr. pomocou štandardného príkazu *ls*, ako môžeme vidieť na nasledujúcom príklade.

```
$ ls -la
total 24
drwxr-xr-x  3 jerry users  4096 Oct  4 18:46 .
drwxrwxrwt 15 root  root  12288 Oct  4 18:46 ..
drwxr-xr-x  2 jerry users  4096 Oct  4 18:46 adresar
-rw-r--r--  1 jerry users     5 Oct  4 18:46 subor
```

typ
práva
vlastník
skupina

Prístupové práva sú používateľovi typicky prezentované ako 9 znakový reťazec zo znakov r, w, x, -. Prvá trojica znakov reprezentuje práva pre vlastníka, druhá pre skupinu a posledná práva pre ostatných. Prvý znak v trojici (r) určuje, či subjekt má právo na čítanie, druhý (w) právo na zápis, a tretí (x) právo na spúšťanie / použitie v ceste. Ak subjekt právo nemá, namiesto r, w, alebo x je vo výpise znak -.

Keďže práva pre jeden subjekt pozostávajú z 3 bitov, je logické ich vyjadrovať aj číselne v osmičkovej sústave – právo pre jeden subjekt sa takto dá vyjadriť ako 1 číslica 0 – 7. V tomto vyjadrení je právo na čítanie reprezentované hodnotou 4, právo na zápis hodnotou 2 a právo na spúšťanie súboru / použitie adresára v ceste hodnotou 1. Kombinácia týchto práv je reprezentovaná súčtom hodnôt reprezentujúcich jednotlivé práva.

Prístupové práva potom vyjadrujeme ako 3-ciferné číslo v osmičkovej sústave – najvýznamnejšia (prvá zľava) číslica vyjadruje práva pre vlastníka, stredná práva pre skupinu a najmenej významná

(posledná zľava) práva pre ostatných. Napr.:

- 640 = rw- r-- ---
  - čítanie a zápis pre vlastníka (4+2), čítanie pre skupinu a nič pre ostatných
- 751 = rwx r-x –x
  - všetko pre vlastníka, čítanie a spúšťanie / použitie adresára pre skupinu a len spúštanie / použitie adresára pre ostatných
- 777 = rwx rwx rwx
  - všetko pre všetkých
- 000 = --- --- ---
  - žiadne práva (pre nikoho)

1.1.2. Zmena vlastníka, skupiny a prístupových práv objektu

Prístupové práva k objektu môže meniť len jeho vlastník alebo root. Presnejšie vyjadrené, prístupové práva môže zmeniť len proces s FSUID = UID vlastníka alebo s FSUID = 0. Na zmenu prístupových práv slúži štandardný príkaz *chmod*. Prvou možnosťou je špecifikovať práva číselne (ako 3-ciferné číslo v osmičkovej sústave) – príkaz

```
chmod práva objekt ...
```

nastaví uvedené práva uvedeným objektom. Druhou možnosťou je špecifikovať postupnosť operácií na modifikáciu existujúcich prístupových práv pre objekt(y):

```
chmod KomuOpPráva[,...] objekt ...
```

Jednotlivé operácie sú oddelené čiarkami (pozor, nesmú tam byť žiadne medzery) a vykonajú sa v uvedenom poradí. Každá operácia pozostáva z trojice (*komu*, *operátor*, *práva*), kde

- *komu* je jedno alebo viac písmen z množiny **u, g, o, a**
  - **u** – vlastník (**u**ser),
  - **g** – skupina (**g**roup),
  - **o** – ostatní (**o**thers),
  - **a** – všetci (**a**ll) – je to presne to isté ako **ugo**,
- *operátor* je znak z množiny +, -, =

- + pridať práva k existujúcim,
- - odobrať práva z existujúcich,
- = nastaviť práva bez ohľadu na existujúce (t.j. nahradiť existujúce)

- *práva* je nula alebo viac písmen z množiny **r, w, x, X**
  - **r, w, x** – predstavujú práva na čítanie, zápis a spúšťanie / použitie adresára v ceste,
  - **X** – znamená **x**, ak je objekt adresárom alebo ak naň už niekto má právo na spúšťanie, inak sa ignoruje.

Pri oboch spôsoboch použitia príkazu *chmod* je možné použitím prepínača *-R* aplikovať práva rekurzívne na celý podstrom, ak je objektom adresár. Použitie príkazu *chmod* si môžeme demonštrovať na nasledujúcich príkladoch:

```
$ chmod 600 a b
$ chmod u=rwx,go= d
$ ls -l
total 12
-rw------- 1 jerry jerry    6 Oct 13 10:01 a
-rw------- 1 jerry jerry    6 Oct 13 10:02 b
drwx------ 2 jerry jerry 4096 Oct 13 10:02 d
$ chmod u+x b
$ ls -l
total 12
-rw------- 1 jerry jerry    6 Oct 13 10:01 a
-rwx------ 1 jerry jerry    6 Oct 13 10:02 b
drwx------ 2 jerry jerry 4096 Oct 13 10:02 d
$ chmod go+X *
$ ls -l
total 12
-rw------- 1 jerry jerry    6 Oct 13 10:01 a
-rwx--x--x 1 jerry jerry    6 Oct 13 10:02 b
drwx--x--x 2 jerry jerry 4096 Oct 13 10:02 d
```

V prvom príklade nastavíme na súboroch *a* a *b* práva pre vlastníka na čítanie a zápis (4+2) a žiadne práva (0) pre skupinu a ostatných a druhým príkazom plné práva (rwx) pre vlastníka a žiadne práva pre skupinu a ostatných na adresár *d*. V ďalšom príklade pridáme na súbore *b* právo na spúšťanie pre vlastníka. Napokon pridáme skupine a ostatným právo na spúšťanie / použitie adresárov na všetky objekty, ktoré sú buď adresárom alebo na ne niekto má právo na spúšťanie. Ako vidieť, dotkne sa to adresára *d* a súboru *b*, ktorý už mal nastavené právo na spúšťanie pre vlastníka, ale nedotkne sa to súboru *a*, ktorý žiadne právo na spúšťanie nastavené pre nikoho nemal.

Zmenu vlastníka objektu môže vykonať štandardne iba root. Jedná sa o citlivú operáciu, keďže

informácia o vlastníkovi sa môže využívať aj na priraďovanie zodpovednosti za obsah súborov. Na zmenu vlastníka slúži príkaz *chown*:

```
chown [-R] vlastník[:skupina] objekt ...
```

Vlastník (a prípadne skupina) môžu byť špecifikovaní buď číselne alebo menom používateľa (skupiny). Prepínač *-R* slúži na zmenu vlastníka rekurzívne v celom podstrome.

Na zmenu skupiny slúži príkaz *chgrp*:

```
chgrp [-R] skupina objekt ...
```

Skupina môže byť špecifikovaná číselne alebo meno, operáciu je možné tiež aplikovať rekurzívne na celý podstrom. Skupinu objektu môže zmeniť root alebo vlastník objektu. Vlastník môže zmeniť skupinu len na takú skupinu, ktorej je členom. To umožňuje používateľovi, ktorý je členom viacerých skupín určovať, akú skupinu majú mať nastavenú jednotlivé jeho objekty, a teda ktorá skupina má mať špecificky určené práva k jednotlivým objektom. Root môže zmeniť skupinu objektu na ľubovoľnú.

1.1.3. Atribúty nového objektu pri jeho vytvorení

Pri vytvorení nového objektu súborového systému je potrebné inicializovať aj jeho atribúty používané pre riadenie prístupu. Vlastník objektu je nastavený na FSUID vytvárajúceho procesu. Skupina objektu je nastavená buď na FSGID vytvárajúceho procesu alebo je nastavená na skupinu adresára, v ktorom je objekt vytvorený (podrobnejšie neskôr).

Prístupové práva sú vypočítané tak, že z hodnoty, ktorú požaduje vytvárajúci proces, sú odstránené práva uvedené v atribúte *umask* vytvárajúceho procesu. Atribút *umask* sa štandardne dedí z procesu na jeho potomkov, a pokiaľ ho teda proces úmyselne neupraví, predstavuje prostriedok, ako ovplyvňovať bezpečnosť nových objektov. Umožňuje totiž definovať, aké práva **nemajú byť** nastavené na nových objektoch. Bežné hodnoty *umask* sú napr.:

- 022 – skupine a ostatným odstráni právo na zápis,
- 077 – skupine a ostatným odstráni všetky práva.

Bežné procesy pri vytváraní bežných súborov typicky požadujú práva 666 (rw pre všetkých) a pri vytváraní bežných adresárov práva 777 (rwx pre všetkých). Aplikáciou *umask* napr. s hodnotou 077 takto dostaneme výsledné práva 600 pre súbory a 700 pre adresáre. Aplikáciou *umask* s hodnotou 022 by sme dostali výsledné práva 644 pre súbory a 755 pre adresáre (čiže skupina a ostatní by mali odstránené právo na zápis). Ak vytvárajúci proces má požiadavky na striktnejšie nastavenie

prístupových práv (napr. pri vytváraní súbory, ktorý bude obsahovať citlivú informáciu), *umask* nijakým spôsobom nebráni nastaveniu striktnejších práv. Napr. ak proces bude požadovať práva 600, tak pri oboch spomenutých hodnotách *umask* by výsledné práva boli 600.

Hodnotu atribútu *umask* pre shell je možné nastaviť použitím príkazu *umask*, napr.:

```
umask 022
```

Takýto príkaz môže byť vhodné umiestniť si napr. do skriptu, ktorý sa vykonáva pri prihlásení používateľa.

1.2. Bity set-UID, set-GID a sticky bit

Pre správnu a bezpečnú funkciu niektorých programov je potrebné, aby proces, v rámci ktorého je program vykonávaný, mal iné oprávnenia ako používateľ, ktorý ho spustil. Typickým príkladom je program na zmenu hesla (*passwd*), ktorý potrebuje vedieť čítať a zapisovať do súboru, v ktorom sú uložené heslá používateľov (*/etc/shadow*). Tento súbor je, samozrejme, prístupný iba root-ovi.

Linux poskytuje dva mechanizmy, pomocou ktorých sa podobné problémy riešia. Prvým je *set-UID* bit v prístupových právach. Ak ho má spustiteľný súbor nastavený v prístupových právach, po jeho úspešnom spustení sa efektívne UID, FSUID a uložené UID procesu nastavia na UID vlastníka súboru. To znamená, že takýto proces bude vykonávať operácie v mene iného používateľa než toho, ktorý ho spustil. Napr. vlastníkom súboru */usr/bin/passwd* je root a tento súbor má nastavený *set-UID* bit. Keď ho používateľ spustí, proces bude operácie vykonávať v mene root-a, a teda bude mať práva na čítanie a zápis do */etc/shadow*.

Druhým, podobným mechanizmom je *set-GID* bit. Ak má spustiteľný súbor nastavený *set-GID* bit v prístupových právach, po jeho úspešnom spustení sa efektívne GID, FSGID a uložené GID nastavia na GID skupiny súboru. To znamená, že namiesto primárnej skupiny používateľa, ktorý program spustil, sa pre riadenie prístupu bude používať skupina spustiteľného súboru, čím proces získa práva pridelené tejto skupine.

Treba podotknúť, že *set-UID* a *set-GID* bity nie je možné použiť v prípade skriptov – dôvodom je bezpečnosť, nakoľko skripty sú často ovplyvnené mnohými vonkajšími parametrami (napr. premennými prostredia), čo umožňuje značné zásahy do ich správania. Preto nie je bezpečné im dávať vyššie práva než má používateľ zodpovedný za ich spustenie.

Pri uvažovaní o využití *set-UID* a *set-GID* mechanizmov je potrebné si uvedomiť, že sú týmto spôsobom poskytujeme programom možnosti, ktoré by za iných okolností nemali. Ak tieto programy obsahujú zneužiteľné chyby, môže to mať značný dopad na systém. Je preto potrebné s

týmito mechanizmami zaobchádzať veľmi opatrne. Programy musia byť správne navrhnuté na použitie s týmito mechanizmami. Napr. ak čítajú alebo zapisujú do súborov určených používateľom, je veľmi dôležité, aby v správny okamih zmenili svoje efektívne alebo súborové UID / GID, inak by mohli prečítať alebo prepísať obsah, ku ktorému používateľ normálne nemá prístup.

Pri výbere vhodného mechanizmu je potrebné použiť prístup najmenších oprávnení. Ak potrebujeme, aby program získal len práva na čítanie alebo zápis do nejakých súborov, ku ktorým používatelia nemajú prístup, väčšinou bude stačiť použitie *set-GID* mechanizmu. Ten je najslabší, pretože nedáva procesu žiadne iné možnosti ako pristupovať k súborom s právami skupiny.

Silnejším mechanizmom je *set-UID* na programe vlastnenom špeciálnym, na tento účel určeným používateľom. Oproti *set-GID* získa navyše práva na zmenu prístupových práv a skupín objektov vlastnených týmto používateľom. Ak proces vytvorí nové objekty, nebude možné zistiť, ktorý skutočný používateľ to spôsobil (v prípade *set-GID* by bol ich vlastníkom skutočný používateľ).

Najsilnejším variantom je použitie *set-UID* na programe vlastnenom root-om – v tomto prípade má proces (základnými prostriedkami) prakticky neobmedzené práva v systéme.

Set-GID bit má význam aj na adresároch. Ak je nastavený, tak pri vytvorení nového objektu v tomto adresári bude skupina tohto objektu inicializovaná na skupinu tohto adresára; ak nie je nastavený, bude skupina objektu inicializovaná na FSGID procesu. Ak je bit nastavený a v adresári vytvoríme podadresár, tak tento podadresár bude mať okrem nastavenie skupiny zároveň aj automaticky tiež nastavený *set-GID* bit. Vďaka tomuto mechanizmu je možné zabezpečiť, že všetky novo vytvorené objekty v zdieľanom adresári slúžiacom napr. pre potreby nejakého tímového projektu budú mať nastavenú správnu skupinu (slúžiacu pre projekt) bez ohľadu na to, ktorý používateľ (člen tímu) objekt vytvoril a akú má tento používateľ primárnu skupinu.

Na adresároch má význam ešte jeden bit – *sticky bit*. Ak je tento bit nastavený na adresári, tak objekt z tohto adresára môže vymazať alebo premenovať len vlastník objektu, vlastník adresára alebo root. Samozrejme, na vymazanie alebo premenovanie objektu sú stále potrebné aj práva na zápis do adresára a na použitie všetkých potrebných adresárov v ceste, čiže len pridudnú ďalšie obmedzenia. Tento mechanizmus sa typicky používa napr. v zdieľaných adresároch, do ktorých majú plné práva všetci používatelia, aby sa zamedzilo vzájomnému vymazávaniu, či premenovávaniu súborov používateľmi navzájom. Typickým príkladom je adresár */tmp*. Bez tohto mechanizmu by každý používateľ mohol vymazať alebo premenovať ľubovoľný objekt v tomto adresári, aj keď nemá žiadny prístup k obsahu tohto objektu.

Na nastavenie *set-UID*, *set-GID* a *sticky* bitov slúži príkaz *chmod*. V číselnom tvare sa tieto bity

vyjadrujú spoločne jednou číslicou v osmičkovej sústave. Táto číslica bude na prvom mieste zľava, za ňou nasledujú číslice vyjadrujúce práva pre vlastníka, skupinu a ostatných. *Set-UID* bit má hodnotu 4, *set-GID* bit má hodnotu 2 a *sticky* bit má hodnotu 1. Je možné ich nastavovať aj pomocou symbolickej manipilácie s právami, v takom prípade sa *set-UID* bit nastavuje ako právo **s** pre vlastníka, *set-GID* bit ako právo **s** pre skupinu a *sticky* bit ako právo **t** (pre ostatných, ale keďže je to jediné možné použitie **t**, tak to nie je potrebné uvádzať).

Príklady nastavenia *set-UID* bitu:

```
chmod u+s súbor

chmod 4755 súbor
```

Príklady nastavenia *set-GID* bitu:

```
chmod g+s súbor

chmod 2755 súbor
```

Príklady nastavenia *sticky* bitu:

```
chmod 1777 adresár

chmod +t adresár
```

Pri zobrazení prístupových práv vo výstupe príkazu *ls* sa tieto bity zobrazujú ako symbol **s** alebo **t** na pozícii práva pre spúšťanie / použitie adresára. Ak v právach nie je nastavené príslušné právo na spušťanie / použitie adresára, tak sa tieto bity zobrazia ako veľké **S** alebo **T**.

```
$ ls -la /usr/bin/passwd
-rwsr-xr-x 1 root root 34740 Feb 15  2011 /usr/bin/passwd

$ ls -la /usr/bin/crontab
-rwxr-sr-x 1 root crontab 30248 Dec 19  2010 /usr/bin/crontab

$ ls -lad /tmp
drwxrwxrwt 15 root root 12288 Oct  4 20:17 /tmp
```

Význam *set-GID* bitu na adresároch môžeme demonštrovať na nasledujúcom príklade:

```
$ mkdir projekt
$ chgrp projekt projekt
$ chmod g+s projekt
$ touch projekt/subor
$ mkdir projekt/adresar
```

```
$ ls -la projekt
total 12
drwxr-sr-x 3 jerry projekt 4096 Oct  4 19:51 .
drwxr-xr-x 3 jerry jerry   4096 Oct  4 19:50 ..
drwxr-sr-x 2 jerry projekt 4096 Oct  4 19:51 adresar
-rw-r--r-- 1 jerry projekt    0 Oct  4 19:51 subor
```

1.3. Možnosť zmeny UID / GID procesu

Procesy môžu počas svojej práce zmeniť svoje UID a GID nasledovným spôsobom:

- Ak je efektívne UID = 0, proces si môže nastaviť všetky UID, GID a zoznam doplnkových skupín na ľubovoľné hodnoty-
- Inak proces môže svoje UID / GID nastavovať len na aktuálnu hodnotu niektorého UID / GID.
- Vždy pri zmene efektívneho UID / GID sa automaticky zmení aj FSUID / FSGID. FSUID / FSGID je možné nastaviť aj samostatne.

Najčastejšie procesy menia svoje efektívne UID / GID na reálne alebo uložené. To umožňuje procesu, ktorý získal oprávnenia pomocou *set-UID* alebo *set-GID* mechanizmu prepínať medzi využívaním získaných oprávnení a oprávnení používateľa, ktorý tento program spustil. Proces, ktorého efektívne UID je root (0), môže svoju identitu zmeniť ľubovoľne, no akonáhle si všetky UID nastaví na nenulovú hodnotu, už nebude môcť získať oprávnenia root-a naspäť.

2. Doplnkové atribúty *append-only* a *immutable*

OS Linux umožňuje pri použití bežných Linuxových súborových systémov každému objektu nastaviť doplnkové atribúty, z ktorých dva majú význam z hľadiska bezpečnosti.

Ak má objekt nastavený atribút *append-only*, nie je možné ho vymazať, premenovať, zmeniť jeho vlastníka, skupinu, prístupové práva k nemu, ani naň vytvoriť hard-link. Ak je takýmto objektom súbor, nie je možné do neho zapisovať inak, než pridávať nový obsah na koniec (súbor musí byť otvorený v tzv. *append* móde). Ak je takýmto objektom adresár, je možné v tomto adresári vytvárať nové objekty, ale nie je možné vymazať ani premenovať existujúce. Typické použitie *append-only* atribútu je napr. na log-súbory, ktoré týmto spôsobom môžeme chrániť proti zmene zaznamenaných udalostí.

Ak má objekt nastavený atribút *immutable*, nie je možné ho modifikovať, ani ho vymazať, premenovať, zmeniť jeho vlastníka, skupinu, prístupové práva k nemu, či vytvoriť naň hard-link.

Uvedené obmedzenia sa vzťahujú na všetky procesy vrátane tých, ktoré vykonávajú operácie v mene root-a. Avšak procesy s právami root-a môžu (ak tomu nebránia pokročilé prostriedky, ktorým sa budeme venovať neskôr) tieto atribúty meniť. V každom prípade je použitie týchto atribútov vhodné ako ochrana proti manipulácii zo strany bežných používateľov, ako aj proti neúmyselnej manipulácii zo strany root-a (a procesov s právami root-a).

Aktuálne doplnkové atribúty objektu je možné zobraziť pomocou príkazu *lsattr*:

```
lsattr [-R] [objekt ...]
```

Parametre tohto príkazu sú podobné, ako pri použití príkazu *ls*. Bez parametrov príkaz zobrazí zoznam objektov v aktuálnom adresári a ich doplnkové atribúty. Ak uvedieme meno objektu, tak zobrazí atribúty uvedeného objektu (alebo viacerých); ak je uvedený objekt adresárom, tak zobrazí zoznam a atribúty objektov v tomto adresári. Pomocou prepínača *-R* môžeme rekurzívne zobraziť atribúty všetkých objektov v celom podstrome. Ak chceme zobraziť atribúty adresára (a nie objektov v ňom), môžeme použiť prepínač *-d*:

```
lsattr -d adresár
```

Atribút *append-only* je vo výstupe reprezentovaný znakom **a**, atribút *immutable* znakom **i**.

Na zmenu doplnkových atribútov slúži príkaz *chattr*:

```
chattr [-R] OpAtribúty objekt ...
```

Zmena atribútov je špecifikovaná operáciou a zoznamom atribútov, s ktorými sa má operácia

vykonať. Operácia je určená znakom +, -, alebo =:

- + pridať atribút k existujúcim,
- - odobrať atribút od existujúcich,
- = vymazať existujúce atribúty a nahradiť ich uvedenými.

Zoznam atribútov pozostáva zo jedného alebo viacerých znakov reprezentujúcich atribúty (**i** – *immutable*, **a** – *append-only*). Príkaz je možné aplikovať aj rekurzívne (*-R*) na celý podstrom.

Použitie týchto príkazov môžeme demonštrovať na nasledujúcom príklade:

```
# mkdir adresar
# touch subor
# chattr +a adresar subor
# lsattr
-----a------------- ./adresar
-----a------------- ./subor
# echo 1 > subor
bash: subor: Operation not permitted
# echo 1 >> subor
# rm subor
rm: cannot remove `subor': Operation not permitted
# mv subor subor2
mv: cannot move `subor' to `subor2': Operation not permitted
# chmod 777 subor
chmod: changing permissions of `subor': Operation not permitted
# chown root subor
chown: changing ownership of `subor': Operation not permitted
# ln subor subor2
ln: creating hard link `subor2' => `subor': Operation not permitted
# touch adresar/subor
# mkdir adresar/podadresar
# mv adresar/subor adresar/subor2
mv: cannot move `adresar/subor' to `adresar/subor2': Operation not
permitted
# rm adresar/subor
rm: cannot remove `adresar/subor': Operation not permitted
# echo 1 > adresar/subor
# touch adresar/podadresar/subor
# rm adresar/podadresar/subor
# chmod 666 adresar/subor
# chattr =i adresar subor
# lsattr
----i-------------- ./adresar
----i-------------- ./subor
# rm adresar/subor
rm: cannot remove `adresar/subor': Permission denied
# touch adresar/novy
```

```
touch: cannot touch `adresar/novy': Permission denied
# rm subor
rm: cannot remove `subor': Operation not permitted
# chmod 666 subor
chmod: changing permissions of `subor': Operation not permitted
# echo 1 >> subor
bash: subor: Permission denied
```

3. Rozšírenie ACL (Access Control List)

Základné prístupové práva v OS Linux majú niekoľko nedostatkov, ktoré sa prejavia v prípadoch, keď máme komplikovanejšie požiadavky na nastavenie oprávnení rôznych používateľov či skupín. Prvým problémom je príliš hrubé členenie subjektov, ktorým je možné určiť špecifické prístupové práva. Na to, aby sme dali prístupové práva k objektu niekoľkým používateľom, je potrebné vytvoriť skupinu, ktorej budú členmi a prideliť práva tejto skupine. Problém nastane, ak budeme potrebovať, aby napr. 2 skupiny mali rôzne prístupové práva k objektu.

Druhým problémom je, že nie je možné presne definovať prístupové práva pre nové objekty v adresári. Je možné vynútiť skupinu (použitím *set-GID* bitu), ale prístupové práva je možné len čiastočne ovplyvniť pomocou *umask*. Ak však potrebujeme rôzne počiatočné prístupové práva v rôznych adresároch, museli by si používatelia meniť *umask* podľa toho, v ktorom adresári sa chystajú vytvoriť súbor. Navyše, ak si *umask* správne nenastavia, môžu napr. vytvoriť súbory, ktoré nebudú prístupné používateľom, ktorým byť prístupné majú, alebo naopak, budú prístupné aj tým, ktorým prístupné byť nemajú.

Odpoveďou na tieto nedostatky je rozšírenie prístupových práv známe ako *ACL* (*Access Control List*). ACL nemení existujúce prístupové práva (r, w, x), ale rozširuje množinu typov subjektov, ktorým je možné práva prideľovať. Taktiež umožňuje definovať pre každý adresár samostatne, aké prístupové práva majú mať nové objekty pri vytvorení. Pomocou ACL je možné prístupové práva k objektu súborového systému nastaviť pre:

- vlastníka,
- konkrétnych používateľov,
- skupinu objektu,
- konkrétne skupiny,
- ostatných.

Zeleným uvedené typy subjektov predstavujú rozšírenie oproti základnému modelu prístupových práv.

3.1. ACL pre objekty súborového systému

Ku každému objektu súborového systému môže byť priradený zoznam položiek, ktorého každá položka obsahuje:

- typ položky,
- identifikátor skupiny alebo používateľa,
- prístupové práva (3 bity, r, w, x).

ACL podporujú nasledujúce typy položiek:

- ACL_USER_OBJ – určuje práva pre vlastníka
- ACL_USER – určuje práva pre určeného používateľa
- ACL_GROUP_OBJ – určuje práva pre skupinu objektu
- ACL_GROUP – určuje práva pre určenú skupinu
- ACL_OTHER – určuje práva pre ostatných
- ACL_MASK – určuje maximálne práva pre položky typu ACL_USER, ACL_GROUP a ACL_GROUP_OBJ

ACL obsahuje práve jednu položku každého červeného typu, môže obsahovať 0 alebo viac položiek každého zeleného typu. Ak ACL obsahuje aspoň jednu položku zeleného typu, musí obsahovať práve jednu položku typu ACL_MASK, inak položku typu ACL_MASK obsahovať nemusí.

Pri použití ACL sa algoritmus vyhodnotenia prístupových práv upravuje nasledovne:

- Ak je FSUID procesu rovné UID vlastníka objektu, použijú sa práva z položky ACL_USER_OBJ.
- Inak, ak je FSUID procesu rovné identifikátoru používateľa v niektorej položke typu ACL_USER, použijú sa práva z tejto položky po úprave operáciou *AND* s právami uvedenými v položke typu ACL_MASK. Inými slovami, použijú sa práva pre konkrétneho používateľa, ale maximálne práva uvedené v ACL_MASK.
- Inak sa nájdu postupne všetky položky typu ACL_GROUP_OBJ a ACL_GROUP, ktoré zodpovedajú FSGID procesu alebo GID niektorej skupine zo zoznamu doplnkových skupín procesu (pri položkách typu ACL_GROUP_OBJ sa porovnávajú so skupinou procesu, pri položkách typu ACL_GROUP so skupinou uvedenou v položke). Ak zodpovedajúca

položka obsahuje všetky požadované práva, práva uvedené v položke sa upravia operáciou *AND* s právami uvedenými v položke typu ACL_MASK a výsledné práva sa použijú. Ak zodpovedajúca položka neobsahuje všetky požadované práva, skontroluje sa ďalšia zodpovedajúca položka. Ak existuje aspoň jedna zodpovedajúca položka, no žiadna neobsahuje všetky potrebné práva, operácia sa zamietne.

- Ak sa nenájde žiadna zodpovedajúca položka (t.j. neexistuje žiadna položka pre niektorú skupinu procesu), použijú sa práva z položky ACL_OTHER.
- Nakoniec sa overí, či podľa vyššie uvedeného postupu zvolené práva obsahujú všetky potrebné práva na vykonanie operácie – ak áno, operácia sa povolí, ak nie, operácia sa zamietne.

Ak ACL neobsahuje žiadne položky typu ACL_USER alebo ACL_GROUP, uvedený algoritmus zodpovedá pôvodnému algoritmu používanému bez ACL s tým, že položka typu ACL_USER_OBJ zodpovedá základným právam pre vlastníka, položka typu ACL_GROUP_OBJ zodpovedá základným právam pre skupinu a položka typu ACL_OTHER zodpovedá základným právam pre ostatných.

Skutočne, základné prístupové práva pre vlastníka, skupinu a ostatných zodpovedajú niektorým položkám ACL. Ak dôjde k zmene základných prístupových práv (napr. pomocou *chmod*), táto zmena sa premietne aj do zmeny príslušnej položky ACL. Naopak, ak sa zmení príslušná položka v ACL, táto zmena sa premietne aj do zmeny základných prístupových práv. Tým je zabezpečná spätná kompatibilita s programami, ktoré nepoužívajú ACL. Vzťah medzi základnými právami a položkami ACL je však o niečo komplikovanejší. Základné práva pre vlastníka vždy zodpovedajú položke typu ACL_USER_OBJ a základné práva pre ostatných vždy zodpovedajú položke typu ACL_OTHER. Ak ACL neobsahuje položku typu ACL_MASK (a teda ani typu ACL_USER či ACL_GROUP), tak základné práva pre skupinu zodpovedajú položke ACL_GROUP_OBJ. Ak ale ACL obsahuje položku typu ACL_MASK, tak základné práva pre skupinu zodpovedajú práve tejto položke. Tento posledný fakt môže byť na prvý pohľad kontraintuitívny, má však opodstatnenie. Dôvodom je kompatibilita s programami, ktoré s existenciou ACL príliš nepočítajú, no ktoré potrebujú nastaviť alebo overiť prístupové práva napr. tak, aby mal k súboru prístup len jeho vlastník. Takéto programy predpokladajú, že ak sú práva nastavené na 600, nikto okrem vlastníka k súboru prístup nemá. Ak však tomuto súboru priradíme ACL, môžeme v ňom definovať práva napr. pre ďalších používateľov alebo skupiny. Keďže ale v takom prípade základné práva pre skupinu (0) zodpovedajú položke typu ACL_MASK, žiadne práva pridelené ďalším používateľom alebo

skupinám nebudú efektívne uplatniteľné, keďže po operácii *AND* s hodnotou 0 budú rovné 0. Preto sa práva pre skupiny a práva pre explicitne určených používateľov upravujú operáciou *AND* s právami v maske.

V jadre OS Linux (v mnohých verziách vrátane aktuálnych v čase písania tohto textu) je však chyba, ktorá spôsobuje, že ak sú základné práva pre skupinu nulové, tak sa ACL ignoruje. Ak ACL neobsahuje položku typu ACL_MASK, táto chyba sa prakticky neprejaví, pretože použitie ACL v takom prípade je ekvivalentné štandardnému algoritmu. Avšak v prípade, že ACL obsahuje položku typu ACL_MASK s nulovými právami a zároveň obsahuje položku typu ACL_GROUP zodpovedajúcu niektorej skupine procesu, výsledok by mal byť zamietnutie operácie (keďže by boli použité nulové práva). Ale keďže sa v dôsledku tejto chyby nepoužije ACL, v skutočnosti sa použijú práva pre ostatných. A ak práva pre ostatných nie sú tiež nulové, tak operácia môže byť povolená. Našťastie takáto kombinácia nastavení nie je veľmi pravdepodobná – nie je typické, že by ostatní mali mať menšie práva ako nejaká skupina, ktorá má explicitne určené práva. V každom prípade je to chyba, o ktorej je dobre vedieť.

3.1.1. Zobrazenie a nastavenie ACL

Na zobrazenie a nastavenie ACL sa používajú dve textové reprezentácie – dlhá a krátka. V dlhej textovej reprezentácii je každá položka ACL uvedená na samostatnom riadku a pozostáva z troch políčok oddelených dvojbodkou:

```
typ:id:práva
```

Typ môže byť

- *user* – položka typu ACL_USER alebo ACL_USER_OBJ (ak je *id* prázdne),
- *group* – položka typu ACL_GROUP alebo ACL_GROUP_OBJ (ak je *id* prázdne),
- *mask* – položka typu ACL_MASK,
- *other* – položka typu ACL_OTHER.

Práva pozostávajú z 3 znakov z množiny **r, w, x, -**. Sú uvedené vždy v poradí **rwx** s tým, že namiesto písmen je uvedená pomlčka, ak príslušné právo nie je nastavené.

Krátka textová reprezentácia obsahuje všetky položky v jednom riadku, oddelené čiarkami. Typ môže byť skrátený na prvé písmeno (t.j. **u, g, m, o**) a pomlčky v právach sa môžu vynechať, ale práva musia obsahovať aspoň jeden znak (prázdne práva musia obsahovať aspoň jednu pomlčku).

ACL môžeme zobraziť pomocou príkazu *getfacl*:

```
getfacl [-R] objekt
```

Príkaz vypíše ACL uvedeného objektu v dlhej textovej reprezentácii doplnenej o komentáre s informáciou o mene objektu, jeho vlastníkovi a skupine. Pomocou prepínača *-R* môžeme zobraziť ACL rekurzívne pre celý podstrom.

Použitie si môžeme demonštrovať na nasledujúcom príklade:

```
# umask
0022
# mkdir adresar
# ls -lad adresar
drwxr-xr-x 2 root root 4096 Oct  5 13:58 adresar

# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
group::r-x
other::r-x
```

Na nastavenie ACL slúži príkaz *setfacl*. Pomocou prepínača *-R* je ho možné tiež aplikovať rekurzívne na celý podstrom. Prvým typom operácie, ktorú je možné s ACL vykonať, je pridanie alebo modifikácia položky ACL:

```
setfacl [-R] -m acl objekt
```

- *acl* – popis položiek ACL, ktoré sa majú pridať alebo upraviť v krátkej textovej reprezentácii.

Ak príslušná položka, určená typom a identifikátorom používateľa alebo skupiny v ACL už existuje, tak sa jej práva zmenia, ak neexistuje, tak sa vytvorí.

Na odstránenie položiek z ACL môžeme použiť prepínač *-x*:

```
setfacl [-R] -x acl objekt
```

- *acl* – popis položiek ACL, ktoré sa majú odstrániť v krátkej textovej reprezentácii bez uvedenia práv (teda len *typ:id*).

Na odstránenie celého ACL slúži prepínač *-b*:

```
setfacl [-R] -b objekt
```

Ak v ACL zostanú len položky typu ACL_USER_OBJ, ACL_GROUP_OBJ a ACL_OTHER, tak

ACL bude tiež automaticky odstránený.

Niekedy môže byť užitočné mať možnosť exportovať ACL pre celý podstrom do súboru a následne môcť ACL v celom podstrome z tohto súboru obnoviť. Typický prípad, kedy je táto možnosť užitočná, je pri zálohovaní a obnove podstromu, v ktorom sa používa ACL, pomocou nástrojov, ktoré ACL priamo nepodporujú. Analogická situácie je pri prenose podstromu na iný počítač. Na export ACL stačí použiť *getfacl* s prepínačom *-R* a výstup uložiť do súboru. Na následnú obnovu ACL z tohto súboru môžeme použiť *setfacl* s nasledujúcim parametrom:

```
setfacl --restore=súbor
```

Pri tomto použití *setfacl* využíva aj komentáre v tomto súbore na určenie ciest k jednotlivým objektom, ktorým má nastaviť ACL.

Ak pri úprave ACL príkazom *setfacl* nešpecifikujeme explicitne položku typu ACL_MASK, príkaz automaticky vypočíta vhodnú masku práv tak, že skombinuje všetky práva uvedené po úprave ACL v položkách typu ACL_USER, ACL_GROUP a ACL_GROUP_OBJ. Tým sa dosiahne stav, že všetky práva uvedené v týchto položkách budú aj efektívne uplatniteľné, teda nebudú blokované maskou. V mnohých prípadoch je toto správanie vhodné, pretože šetrí potrebu ručne špecifikovať správnu masku. Ak máme však masku úmyselne nastavenú na nejakú hodnotu, ktorá blokuje niektoré práva, takéto automatické prepočítanie masky by nám ju zmenilo. Zabrániť tomu môžeme buď explicitným špecifikovaním masky pri modifikácii ACL, alebo použitím prepínača *-n*, ktorý príkazu *setfacl* zakáže automaticky prepočítať a upraviť masku.

Použitie *setfacl* si môžeme demonštrovať na niekoľkých príkladoh.

```
# setfacl -m u:jerry:rwx,g:users:r,g::-,o::- adresar
# ls -lad adresar
drwxrwx---+ 2 root root 4096 Oct  5 13:58 adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
user:jerry:rwx
group::---
group:users:r--
mask::rwx
other::---
```

Do ACL objektu *adresar* sme pridali položku pre používateľa *jerry* s právami **rwx**, položku pre skupinu *users* s právami **r** a skupine objektu a ostatným sme nastavili práva na prázdne. Za povšimnutie stojí aj výstup príkazu *ls*, v ktorom je prítomnosť ACL indikovaná znakom + za

prístupovými právami. Vo výstupe *getfacl* zas môžeme vidieť, že základné práva pre skupinu už nezodpovedajú položke typu ACL_GROUP_OBJ, ale zodpovedajú položke typu ACL_MASK. Zároveň vidíme, že hoci sme masku nešpecifikovali, automaticky bola vypočítaná ako *OR* všetkých ovplyvňovaných položiek.

Na ďalšom príklade si môžeme demonštrovať správanie ACL pri zmene základných prístupových práv.

```
# chmod g-w adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
user:jerry:rwx          #effective:r-x
group::---
group:users:r--
mask::r-x
other::---
```

Ako si môžeme všimnúť, odobratie základného práva na zápis pre skupinu sa v ACL prejavilo odobratím práva na zápis z masky, čo následne obmedzuje efektívne práva (zobrazené ako komentár vo výstupe *getfacl*) pre položky ovplyvňované maskou.

Na nasledujúcom príklade môžeme vidieť, že automatické prepočítavanie masky sa robí aj pri odstraňovaní položiek z ACL:

```
# setfacl -x g:users adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
user:jerry:rwx
group::---
mask::rwx
other::---
```

Vidíme, že maska sa opäť prepočítala tak, aby zahŕňala všetky ovplyvňované položky, a teda právo na zápis pre používateľa *jerry* je opäť uplatniteľné.

Ak masku uvedieme explicitne, tak sa prepočítavať nebude, ale použije sa nami definovaná hodnota, ako vidieť na nasledujúcom príklade.

```
# setfacl -m g:users:rwx,m::rx adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
user:jerry:rwx          #effective:r-x
group::---
group:users:rwx             #effective:r-x
mask::r-x
other::---
```

Prepočítavaniu masky môžeme zabrániť aj spomínaným prepínačom *-n*:

```
# setfacl -n -x g:users adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
user:jerry:rwx          #effective:r-x
group::---
mask::r-x
other::---
```

Ako vidíme, v tomto prípade pri odstránení položky k prepočítaniu masky nedošlo.

Po odstránení ACL z objektu bude príkaz *getfacl* zobrazovať len položky zodpovedajúce základným prístupovým práam:

```
# setfacl -b adresar
# getfacl adresar
# file: adresar
# owner: root
# group: root
user::rwx
group::---
other::---

# ls -lad adresar
drwx------ 2 root root 4096 Oct  5 13:58 adresar
```

Ako vidieť vo výstupe z *ls*, ACL bol odstránený.

3.2. Default ACL pre adresár

Druhým problémom, ktorý sme spomínali v úvode kapitoly bola nemožnosť definovať predvolené

(*default*) práva novo vznikajúce objekty v konkrétnych adresároch. Rozšírenie ACL rieši aj tento problém tým, že okrem normálne ACL umožňuje pre adresáre definovať aj tzv. *default* ACL. Ak adresár nemá nastavený *default* ACL, tak pri vytvorení nového objektu sa postupuje klasickým spôsobom – skombinujú sa práva požadované vytvárajúcim procesom a *umask*. Ak adresár má nastavený *default* ACL, tak sa klasický postup nepoužije a namiesto neho sa novo vytvorenému objektu nastaví ACL, ktoré vznikne nasledovným postupom:

- Ako základ sa zoberie kópia *default* ACL adresára, v ktorom je objekt vytvorený.
- Z položiek, ktoré zodpovedajú základným prístupovým právam sa odstránia tie práva, ktoré vytvárajúci proces nepožadoval.
- Výsledné ACL sa nastaví ako ACL objektu.
- Ak je novým objektom adresár, tak sa mu zároveň nastaví aj *default* ACL, aby sa inicializácia ACL nových objektov zachovala aj pre objekty vytvorené v novom podadresári.

Na nastavovanie *default* ACL slúži príkaz *setfacl* podobne ako na nastavovanie „normálnych“ ACL. Na pridanie položky alebo zmenu existujúcej položky môžeme príkaz použiť jedným z nasledujúcich dvoch spôsobov:

```
setfacl -d -m acl adresar
```

kde prepínač *-d* určuje, že sa má modifikovať *default* ACL, alebo

```
setfacl -m defacl adresar
```

kde *defacl* obsahuje položky *default* ACL zapísané tak, že obsahujú prefix *default:* alebo *d:*. Týmto spôsobom je možne jedným príkazom nastaviť súčasne aj normálny ACL, aj *default* ACL.

Analogicky na odstránenie položky z *default* ACL môžeme použiť príkazy:

```
setfacl -d -x acl adresar
```

```
setfacl -x defacl adresar
```

Na odstránenie celého *default* ACL môžeme použiť príkaz:

```
setfacl -k adresar
```

Manipuláciu a efekt *default* ACL si môžeme demonštrovať na niekoľkých príkladoch.

```
# setfacl -d -m g:users:rwx,u:jerry:rwx adresar
# getfacl adresar
# file: adresar
# owner: root
```

```
# group: root
user::rwx
group::---
other::---
default:user::rwx
default:user:jerry:rwx
default:group::---
default:group:users:rwx
default:mask::rwx
default:other::---
```

Ako vidíme, aj v prípade *default* ACL funguje automatické vypočítavanie vhodnej masky. Navyše sú do *default* ACL automaticky doplnené aj povinné položky, ak nie sú explicitne určené (preberú sa z normálneho ACL).

Na nasledujúcom príklade môžeme vidieť efekt *default* ACL pri vytvorení nových objektov.

```
# umask
0022
# touch adresar/subor
# mkdir adresar/podadresar
# getfacl adresar/subor
# file: adresar/subor
# owner: root
# group: root
user::rw-
user:jerry:rwx          #effective:rw-
group::---
group:users:rwx             #effective:rw-
mask::rw-
other::---
# getfacl adresar/podadresar/
# file: adresar/podadresar/
# owner: root
# group: root
user::rwx
user:jerry:rwx
group::---
group:users:rwx
mask::rwx
other::---
default:user::rwx
default:user:jerry:rwx
default:group::---
default:group:users:rwx
default:mask::rwx
default:other::---
```

Ako môžeme vidieť, nastavenie *umask* sa neprejavilo (zápisové práva neboli odstránené). Naopak, keďže príkaz *touch* pri vytváraní súboru štandardne požaduje práva 666, boli z masky (položka

zodpovedajúce základným právam pre skupinu) odstránené práva na spúšťanie, hoci v *default* ACL boli prítomné.

3.3. Príklad využitia ACL

Použitie ACL si môžeme teraz demonštrovať na komplexnejšom príklade. Predpokladajme, že chceme vytvoriť adresár *projekt*, s nasledujúcimi vlastnosťami:

- skupina *projekt* má mať plné práva na celý podstrom,
- skupina *staff* má mať práva na čítanie a spúšťanie/použitie adresára,
- používateľ *jerry* má mať plné práva,
- ktokoľvek má mať právo čítať, spúšťať/použiť v podstrome *projekt/pub*.

Začneme vytvorením adresára, nastavením jeho skupiny, nastavením *set-GID* bitu (aby sa skupina *projekt* nastavovala automaticky aj novým objektom) a nastavením ACL a *default* ACL tak, aby sme splnili podmienky zadania:

```
# mkdir projekt
# chgrp projekt projekt
# chmod g+s projekt
# setfacl -m g::rwx,g:staff:rx,u:jerry:rwx,o:x projekt
# setfacl -d -m g::rwx,g:staff:rx,u:jerry:rwx,o:- projekt
# ls -lad projekt
drwxrws--x+ 2 root projekt 4096 Oct  5 17:54 projekt
# getfacl projekt
# file: projekt
# owner: root
# group: projekt
# flags: -s-
user::rwx
user:jerry:rwx
group::rwx
group:staff:r-x
mask::rwx
other::--x
default:user::rwx
default:user:jerry:rwx
default:group::rwx
default:group:staff:r-x
default:mask::rwx
default:other::---
```

Vidíme, že používateľ *jerry*, skupina objektu (*projekt*) a skupina *staff* majú nastavené požadované práva, ostatní majú nulové práva.

Ďalej môžeme vytvoriť podadresár *pub*:

```
# cd projekt; mkdir pub; getfacl pub
# file: pub
# owner: root
# group: projekt
# flags: -s-
user::rwx
user:jerry:rwx
group::rwx
group:staff:r-x
mask::rwx
other::---
default:user::rwx
default:user:jerry:rwx
default:group::rwx
default:group:staff:r-x
default:mask::rwx
default:other::---
```

Čo nám chýba sú práva pre ostatných na adresár *pub* a budúce objekty, ktoré v ňom vzniknú.

```
# setfacl -m o::rx,d:o::rx pub
# getfacl pub
# file: pub
# owner: root
# group: projekt
# flags: -s-
user::rwx
user:jerry:rwx
group::rwx
group:staff:r-x
mask::rwx
other::r-x
default:user::rwx
default:user:jerry:rwx
default:group::rwx
default:group:staff:r-x
default:mask::rwx
default:other::r-x
```

V tomto okamihu máme všetky ACL nastavené správne a ich fungovanie si môžeme demonštrovať vytvorením niekoľkých testovacích súborv a adresárov a preskúmaním ich ACL.

```
# mkdir subdir pub/subdir
# touch file subdir/file pub/file pub/subdir/file
# ls -ld file subdir subdir/file
-rw-rw----+ 1 root projekt    0 Oct  5 18:05 file
drwxrws---+ 2 root projekt 4096 Oct  5 18:05 subdir
-rw-rw----+ 1 root projekt    0 Oct  5 18:05 subdir/file
# ls -ld pub/file pub/subdir pub/subdir/file
```

```
-rw-rw-r--+ 1 root projekt    0 Oct  5 18:05 pub/file
drwxrwsr-x+ 2 root projekt 4096 Oct  5 18:05 pub/subdir
-rw-rw-r--+ 1 root projekt    0 Oct  5 18:05 pub/subdir/file
# getfacl file
# file: file
# owner: root
# group: projekt
user::rw-
user:jerry:rwx          #effective:rw-
group::rwx                   #effective:rw-
group:staff:r-x              #effective:r--
mask::rw-
other::---
# getfacl subdir
# file: subdir
# owner: root
# group: projekt
# flags: -s-
user::rwx
user:jerry:rwx
group::rwx
group:staff:r-x
mask::rwx
other::---
default:user::rwx
default:user:jerry:rwx
default:group::rwx
default:group:staff:r-x
default:mask::rwx
default:other::---
# getfacl pub/file
# file: pub/file
# owner: root
# group: projekt
user::rw-
user:jerry:rwx          #effective:rw-
group::rwx                   #effective:rw-
group:staff:r-x              #effective:r--
mask::rw-
other::r--
# getfacl pub/subdir
# file: pub/subdir
# owner: root
# group: projekt
# flags: -s-
user::rwx
user:jerry:rwx
group::rwx
group:staff:r-x
mask::rwx
other::r-x
```

```
default:user::rwx
default:user:jerry:rwx
default:group::rwx
default:group:staff:r-x
default:mask::rwx
default:other::r-x
```

3.4. Podpora ACL v OS Linux

Na prácu s ACL je potrebné mať inštalovaný balík *acl*. ACL sú ukladané v tzv. rozšírených atribútoch (*extended attributes*) v súborovom systéme, z čoho vyplýva, že môžu byť nastavené len v súborových systémoch, ktoré tieto atribúty podporujú. V súčasnosti sú podporované napr. v súborových systémoch *ext2, ext3, ext4, reiserfs*. Nie sú podporované napr. v súborových systémoch *vfat* alebo *ntfs*.

Aby súborový systém ACL podporoval, je potrebné, aby bol pripojený do stromu s voľbou *acl*, teda príslušný riadok v */etc/fstab* môže vyzerať nasledovne:

```
# <file system> <mount point>   <type>  <options>   <dump>  <pass>
/dev/sda1       /               ext3    acl         0       1
```

4. Subsystém *capabilities*

V tradičnom bezpečnostnom modeli Linux-u aj pri rozšírení o ACL má používateľ root špeciálne postavenie. Na procesy, ktoré vykonávajú súborové operácie v mene root-a, teda procesy s FSUID = 0, sa nevzťahujú obmedzenia vyplývajúce z riadenia prístupu k objektom súborového systému (až na niektoré výnimky ako spúšťanie nespúšťateľných súborov alebo obmedzenia vyplývajúce z doplnkových atribútov *append-only* a *immutable*). Práva root-a sú tiež potrebné na vykonanie rôznych privilegovaných operácií – môže ich vykonať len proces s efektívnym UID = 0. Medzi privilegované operácie patria najmä operácie, ktoré majú globálny vplyv na systém (teda vplyv aj na procesy iných používateľov). Príkladmi typických privilegovaných operácií sú:

- konfigurácia globálnych systémových parametrov (čas, meno počítača, ...),
- pripájanie a odpájanie súborových systémov,
- konfigurácia siete, prístup k nižším vrstvám siete (napr. odpočúvanie sieťovej komunikácie),
- posielanie signálov cudzím procesom,
- ale napr. aj poskytovanie služieb na TCP/UDP portoch nižších ako 1024.

Dôsledkom toho, že privilegované operácie môžu vykonávať len procesy bežiace v mene root-a, je skutočnosť, že v mene root-a beží (aspoň nejaký čas) typicky príliš veľa procesov. Dlhodobo bežiace procesy, ktoré potrebujú vykonať nejakú privilegovanú operáciu, sú typicky spustené v mene root-a a v lepšom prípade neskôr zmenia svoje UID na neprivilegovaného používateľa. Ak však potrebujú privilegované operácie vykonávať opakovane (nie len pri svojom spustení), tak buď svoje UID nemenia, alebo zmenia len svoje efektívne UID, aby mohli neskôr znovu získať práva root-a.

Keď potrebuje privilegované operácie vykonať proces, ktorý spúšťa bežný používateľ, využíva sa *set-UID* mechanizmus. Opäť v lepšom prípade tento proces po vykonaní privilegovanej operácie zmení svoje UID tak, aby už nemohol získať práva root-a. Ak však potrebuje privilegované operácie vykonávať opakovane UID nemení, alebo mení len efektívne UID, tak aby ho opäť mohol zmeniť na root-a.

Ako v prípade dlhodobo bežiacich procesov, tak aj v prípade procesov spúšťaných používateľmi je problémom skutočnosť, že pre jednu privilegovanú operáciu proces získava prakticky neobmedzenú kontrolu nad systémom. Ak program takého procesu obsahuje chyby, tieto môžu mať prakticky neobmedzený dopad na systém. A ak v programe takéhoto procesu existuje zraniteľnosť, ktorá

umožňuje útočníkovi významne ovplyvniť správanie procesu (napr. vložením vlastného vykonateľného kódu), útočník získava prakticky neobmedzené práva.

Uvažujme napríklad program na zachytávanie sieťovej komunikácie. Čítanie prichádzajúcich a odchádzajúcich sieťových paketov je privilegovaná operácia, takže takýto program bude fungovať len za predpokladu, že proces bude mať efektívne práva root-a. To mu však umožní omnoho viac, než len zachytávať sieťovú komunikáciu. Z histórie sú známe chyby v obľúbenom programe na zachytávanie sieťovej komunikácie pre UNIX / Linux (*tcpdump*), ktoré umožňovali útočníkovi vložiť do procesu vlastný kód poslaním vhodne pripraveného paketu. Útočník tak mal možnosť vykonať v systéme prakticky čokoľvek.

Iným pekným príkladom môže byť zálohovanie. Zálohovanie údajov je nepochybne dôležitá a užitočná činnosť. Avšak zálohovanie údajov všetkých používateľov si vyžaduje, aby zálohovací proces mal môžnosť čítať všetky súbory, ktorých obsah má zálohovať. Typické riešenie zálohovania vyzerá v praxi tak, že zálohovací proces je spustený a celý čas beží s právami root-a, aby nebol obmedzený riadením prístupu k súborom. Ak by použitý zálohovací program obsahoval zraniteľnosť zneužiteľnú napr. pomocou vytvorenia súboru s nejak špeciálne zvoleným menom (napr. príliš dlhým, alebo obsahujúcim špeciálne znaky, a pod.), útočník, ktorý by mal možnosť v systéme vytvoriť takýto súbor, by opäť získal neobmedzené prístupové práva.

Ako môžeme vidieť z uvedených príkladov, granularita typu „všetko alebo nič“ vo vzťahu k privilegovaným operáciám naozaj bezpečnosti neprospieva. Subsystém *capabilities* v OS Linux je čiastočným riešením tohto problému. Toto riešenie je založené na dvoch základných princípoch:

- rozdelenie oprávnenia „vykonávať všetky privilegované operácie“ na menšie časti – na oprávnenia vykonávať jednotlivé privilegované operácie alebo malé skupiny operácií,
- prideľovanie oprávnení konkrétnym programom.

Výhodou, ktorú prináša rozdelenie na menšie oprávnenia, je možné významné zníženie dopadu chyby alebo zneužitia zraniteľnosti programu. Ak napr. zálohovaciemu programu umožníme len čítať všetky súbory, bude môcť spôsobiť významne menšie škody (aj keď stále rozhodne nemusia byť zanedbateľné), než keď mu umožníme napr. aj zapisovať do akéhokoľvek súboru. Podobne, keď programu na zachytávanie sieťovej komunikácie umožníme len nízkoúrovňový prístup k sieti, nezíska možnosť čítať alebo zapisovať do akýchkoľvek súborov v systéme.

Výhodou prideľovania oprávnení konkrétnym programom je to, že aj bežný používateľ, ktorý normálne nemôže privilegované operácie vykonať, môže také operácie vykonať prostredníctvom na ten účel určeného programu. Vďaka tomu môžeme napríklad umožniť používateľovi spustiť určený

program na zachytávanie sieťovej komunikácie bez toho, aby sme vytvorili potenciálnu možnosť na získanie plných práv root-a v dôsledku jeho zneužitia. Samozrejme, pomocou metód riadenia prístupu spomenutých v predchádzajúcich kapitolách môžeme zabezpečiť, aby tento program mohli spustiť len vybraní používatelia.

Nevýhodou použitia subsystému *capabilities* je, že na dosiahnutie ideálneho stavu môže byť potrebné upraviť niektoré programy, ktoré majú výhody tohto subsystému naplno využiť. Týka sa to najmä programov, ktoré počítajú s použitím klasických metód ako napr. *set-UID*, a ktoré odmietnu pracovať, ak nebudú explicitne spustené očakávaným spôsobom. Avšak v mnohých prípadoch je využitie subsystému *capabilities* možné bez zásahu do programov, ktoré potrebujú vykonávať niektoré privilegované operácie.

4.1. Vybrané oprávnenia subsystému capabilities

Na úvod sa pozrime na vybrané oprávnenia subsystému *capabilities* a operácie, ktoré pokrývajú. Prvou skupinou oprávnení sú oprávnenia súvisiace riadením prístupu k súborovému systému. Bez použitia subsystému *capabilities* by príslušné operácie mohol vykonať len proces s FSUID = 0.

- CAP_CHOWN
  - oprávnenie na zmenu vlastníka objektu súborového systému,
- CAP_DAC_OVERRIDE
  - oprávnenie na čítanie z / zápis do akéhokoľvek súboru / adresára, použitie akéhokoľvek adresára v ceste, spustenie akéhokoľvek programu, ktorý môže niekto spustiť,
- CAP_DAC_READ_SEARCH
  - oprávnenie na čítanie akéhokoľvek súboru / adresára a použitie adresára v ceste,
- CAP_FOWNER
  - vykonanie operácie, ktorá vyžaduje vlastníctvo objektu (*chmod*, vymazanie / premenovanie objektu v adresári s nastaveným *sticky* bitom),
- CAP_FSETID
  - nastavenie *Set-GID* bitu na súbore cudzej skupiny, nezmazanie *Set-UID* a *Set-GID* pri zápise do súboru,

- CAP_LINUX_IMMUTABLE
    - nastavenie *immutable* a *append-only* atribútu,
- CAP_MKNOD
    - vytváranie blokových a znakových zariadení.

Ďalšou skupinou oprávnení sú oprávnenia na operácie ako zmena identity (nastavenie UID a GID atribútov procesu), posielanie signálov cudím procesom, sledovanie činnosti cudzieho procesu a manipulácia s jeho adresným priestorom, nastavenie vyššej priority procesu, a pod. Tieto operácie by inak vyžadovali, aby proces mal efektívne UID = 0.

- CAP_SETUID
    - nastavenie UID na ľubovoľné hodnoty,
- CAP_SETGID
    - nastavenie GID a doplnkových skupín na ľubovoľné hodnoty,
- CAP_KILL
    - poslanie signálu ľubovoľnému procesu,
- CAP_SETFCAP
    - nastavenie oprávnení (*capabilities*) pre program,
- CAP_SETPCAP
    - niektoré manipulácie s oprávneniami (*capabilities*),
- CAP_SYS_PTRACE
    - použitie *ptrace* (operácia na sledovanie činnosti procesu a na prístup do jeho adresného priestoru) na ľubovoľný proces,
- CAP_SYS_NICE
    - zvýšenie priority procesu.

Ďalšia skupina oprávnení pokrýva privilegované sieťové operácie:

- CAP_NET_ADMIN
  - konfigurácia siete (rozhrania, routovanie, firewall, promiskuitný mód, …),
  - povolenie multicastingu, ...,
- CAP_NET_BIND_SERVICE
  - počúvanie na privilegovaných TCP/UDP portoch (<1024),
- CAP_NET_RAW
  - použitie RAW paketov, zachytávanie sieťovej komunikácie.

Napokon spomeňme ďalšie vybrané oprávnenia, ktoré tradične prináležia len root-ovi:

- CAP_SYS_ADMIN
  - ďalšie administrátorské operácie (napr. mount, privilegované operácie so zariadeniami, …),
- CAP_SYS_BOOT
  - vypnutie a reboot systému,
- CAP_SYS_CHROOT
  - použitie volania *chroot*,
- CAP_SYS_MODULE
  - zavedenie a odstránenie modulu do/z jadra,
- CAP_SYS_RAWIO
  - oprávnenia na I/O operácie,
- CAP_SYS_TIME
  - manipulácia s časom,
- CAP_SYSLOG
  - konfigurácia logovania z kernelu.

4.2. Fungovanie subsystému capabilities

V súčasných verziách jadra OS Linux je vykonanávanie všetkých privilegovaných operácií kontrolované prostredníctvom subsystému *capabilities*. Neskôr si ukážeme, akým spôsobom je štandardne zabezpečená kompatibilita s tradičným bezpečnostným modelom – teda ako štandardne procesy s efektívnym UID (resp. FSUID pri súborových operáciách) získavajú oprávnenia subsystému *capabilities*.

4.2.1. Oprávnenia procesu

Každý proces má priradené štyri množiny (od jadra 4.3 päť množín) oprávnení:

- *permitted* (povolené),
- *inheritable* (zdediteľné),
- *effective* (efektívne),
- *bounding set* (ohraničujúca množina),
- *ambient set* (okolité) [od jadra 4.3].

Pri vyhodnocovaní toho, či proces môže vykonať požadovanú privilegovanú operáciu je rozhodujúci momentálny obsah množiny efektívnych (*effective*) oprávnení. Tá teda predstavuje množinu aktívnych oprávnení. Obsah tejto množiny si proces môže použitím príslušných systémových volaní meniť. Jej obsah musí však vždy byť podmnožinou povolených (*permitted*) oprávnení. To umožňuje procesu, aby si počas svojej činnosti v medziach povolených oprávnení reguloval efektívne (aktívne) oprávnenia. Vďaka tomu si proces môže znížiť svoje oprávnenia v čase, keď ich nepotrebuje, resp. keď je priam nežiadúce, aby oprávnenia mal. V princípe je to podobné tomu, keď si proces využívajúci klasický mechanizmus *set-UID* dočasne zmenil efektívne UID na bežného používateľa – napr. keď potreboval pristupovať k súboru určenému používateľom.

Množina povolených (*permitted*) oprávnení predstavuje teda maximálnu množinu oprávnení, ktoré si proces môže voľne pridávať a odoberať z množiny efektívnych (*effective*) oprávnení. Túto množinu si proces môže upravovať len odoberaním oprávnení. Akonáhle si nejaké oprávnenie z tejto množiny odoberie, už si ho nemôže pridať naspäť, a nemôže ho mať ani v množine efektívnych oprávnení. Opäť je to podobné, ako keď si proces využívajúci klasický *set-UID* mechanizmus zmenil všetky UID na nenulovú hodnotu – už nemal možnosť získať práva root-a.

Množina zdediteľných (*inheritable*) oprávnení určuje, ktoré z oprávnení procesu môžu ostať zachované pri spustení nového programu (čo ešte neznamená, že aj zostanú – k detailom sa

dostaneme neskôr). Proces môže z tejto množiny odstrániť akékoľvek oprávnenie, ale pridať do nej môže len oprávnenie, ktoré má zároveň v množine povolených (*permitted*) oprávnení a v ohraničujúcej množine (*bounding set*). Výnimkou je proces, ktorý má efektívne oprávnenie CAP_SETPCAP; takýto proces si do množiny zdediteľných (*inheritable*) oprávnení môže pridať akékoľvek oprávnenie, ktoré má v ohraničujúcej množine (*bounding set*).

Od jadra 4.3 pribudla množina *ambient set*, ktorá slúži na zachovanie oprávnení pri spustení iného programu, ktorý sám nemá žiadne pridelené oprávnenia a nevyužíva *set-UID* ani *set-GID* mechanizmus. Táto množina musí byť vždy podmnožinou povolených (*permitted*) a zdediteľných (*inheritable*) oprávnení (pri odstránení oprávnenia z niektorej z nich sa automaticky odstráni aj z *ambient*).

Ohraničujúca množina (*bounding set*) predstavuje hornú hranicu oprávnení, ktoré môže proces získať spustením programu, ktorý má pridelené oprávnenia. Zároveň slúži ako horná hranica pre pridávanie oprávnení do množiny zdediteľných (*inheritable*) oprávnení. Bežný proces ohraničujúcu množinu (*bounding set*) modifikovať. Ak má proces efektívne oprávnenie CAP_SETPCAP, tak si môže z ohraničujúcej množiny oprávnenia odoberať. Žiadny proces si však do ohraničujúcej množiny nemôže oprávnenia pridať. Táto vlastnosť umožňuje procesu s oprávnením CAP_SETPCAP vytvoriť podstrom procesov, v ktorom už nebude možné žiadnym spôsobom niektoré oprávnenia získať. Tým je možné napr. zabezpečiť, že žiadny nový proces (v danom podstrome procesov) nebude môcť získať oprávnenie CAP_LINUX_IMMUTABLE, a teda že určite nebude môcť meniť *immutable* a *append-only* atribúty objektov súborového systému. Rovnako je môžné, samozrejme, vylúčiť akékoľvek iné oprávnenie subsystému *capabilities*. To umožňuje veľmi silne obmedziť oprávnenia, ktoré proces (a jeho potomkovia) môžu nadobúdať, a to aj v prípade, že napr. spustia nejaký program (vlastnený root-om) s nastaveným *set-UID* bitom, alebo program, ktorý má pridelené nejaké oprávnenia. Pre úplnosť dodáme, že ohraničujúca množina (*bounding set*) procesu *init* (prvý proces v systéme, koreň stromu procesov) na začiatku obsahuje všetky oprávnenia.

Všetky štyri množiny oprávnení procesu sa dedia z procesu na jeho potomkov (t.j. procesy, ktoré vytvorí). Zmeniť sa môžu buď tak, ako sme uviedli pri popise jednotlivých množín oprávnení, alebo spustením programu (zo spustiteľného súboru).

4.2.2. Oprávnenia pri spustení programu (spustiteľného súboru)

Spustiteľný súbor (program) má priradené dve množiny oprávnení a jeden špeciálny bit:

- *permitted* (povolené) – množina oprávnení,

- *inherited* (zdediteľné) – množina oprávnení,
- *effective* (efektívne) – 1 bit.

Množina povolených (*permitted*) oprávnení programu určuje oprávnenia, ktoré má proces získať spustením tohto programu. Množina zdediteľných (*inheritable*) oprávnení programu určuje, ktoré zdediteľné oprávnenia procesu majú zostať zachované po spustení tohto programu. Bit *effective* (efektívne) určuje, či má byť množina efektívnych (*effective*) oprávnení procesu po spustení tohto programu naplnená všetkými povolenými (*permitted*) oprávneniami, alebo či má byť vyprázdnená.

Ak proces spustí nový program (vykoná operáciu *execve*), jeho oprávnenia sa určia na základe aktuálnych oprávnení procesu a oprávnení spúšťaného programu podľa nasledujúceho algoritmu. Označme

- P(permitted), P(inheritable), P(effective), P(bounding_set) množiny oprávnení procesu pred spustením programu,
- F(permitted), F(inheritable), F(effective) príslušné atribúty programu, a
- P'(permitted), P'(inheritable), P'(effective), P'(bounding_set) množiny oprávnení procesu po spustení programu.

Spustením programu budú hodnoty atribútov procesu transformované nasledovne:

- P'(permitted) = (P(inheritable) & F(inheritable)) | (F(permitted) & P(bounding_set)) | P'(ambient)
- P'(effective) = ak F(effective) = 1, tak P'(permitted), inak P'(ambient)
- P'(inheritable) = P(inheritable)
- P'(ambient) = ak má spúšťaný program nejaké priradené oprávnenie alebo nastavený *set-UID* alebo *set-GID* bit, tak 0, inak P(ambient)
- P'(bounding_set) = P(bounding_set)

Ako vidíme, tak zdediteľné (*inheritable*) oprávnenia procesu a ohraničujúca množina (*bounding set*) sa nezmenia. Pôvodné povolené (*permitted*) a efektívne (*effective*) oprávnenia sa zabudnú a nahradia novými hodnotami. Nové povolené (*permitted*) oprávnenia môže proces získať dvoma spôsobmi (resp. ich kombináciou).

Prvou možnosťou je, že má oprávnenia pridelené spúšťaný program. Tieto sa ohraničia ohraničujúcou množinou (*bounding set*) procesu, t.j. proces nezíska také oprávnenia programu, ktoré nemá vo svojej ohraničujúcej množine (ako sme spomenuli vyššie, vyradením oprávnenia z

ohraničujúcej množiny môžeme zabrániť tomu, aby také oprávnenie mohlo byť neskôr získané).

Druhou možnosťou je získanie oprávnení, ktoré sú v prieniku zdediteľných (*inheritable*) oprávnení procesu a spúšťaného programu. Zaujímavosťou je, že tieto nie sú explicitne ohraničené ohraničujúcou množinou (*bounding set*). Proces si ale nemôže do množiny zdediteľných (*inheritable*) oprávnení pridať nič, čo nemá v ohraničujúcej množine (*bounding set*). Môže však nastať situácia, kedy si privilegovaný proces odstráni z ohraničujúcej množiny oprávnenie, ktoré ešte má v množine zdediteľných oprávnení. Preto, ak proces chce vytvoriť podstrom procesov, kde nebude možné nejaké oprávnenie získať, nesmie ho ponechať ani v množine zdediteľných oprávnení.

Prvý spôsob umožňuje prideliť oprávnenia programu bez ohľadu na to, kto ho spustí (s výnimkou prípadného vyradenia niektorých oprávnení z ohraničujúcej množiny). Predstavuje teda náhradu za klasické riešenie v podobe nastavenia *set-UID* bitu programu vlastnenému root-om.

Druhý spôsob umožňuje prideliť programu oprávnenia „podmienečne“, teda len vtedy, keď ho spustí proces, ktorý má príslušné oprávnenia obsiahnuté v množine zdediteľných. Tento prístup sa dá rozšíriť na prideľovanie oprávnení kombinácii používateľa a programu. Pri prihlásení používateľa je možné pomocou vhodného autentifikačného modulu (pre subsystém PAM) zabezpečiť nastavenie zdediteľných (*inheritable*) oprávnení používateľovmu shell-u. Tie budú následne zdedené všetkými procesmi používateľa a môžu sa teda skombinovať so zdediteľnými oprávneniami programu.

Efektívne (*effective*) oprávnenia procesu po spustení programu závisia od *effective* bitu spusteného programu. Ak je nastavený na 1, tak proces bude mať na začiatku všetky povolené (*permitted*) oprávnenia aj aktívne, ak je nastavený na 0, proces nebude mať na začiatku žiadne aktívne oprávnenia (ale bude si ich môcť pridať), resp. mu zostanú zachované okolité (*ambient*) oprávnenia.

Za povšimnutie stojí skutočnosť, že ak proces nemá nastavené žiadne okolité (*ambient*) oprávnenia a program nemá pridelené žiadne oprávnenia, tak jeho spustením proces stratí všetky oprávnenia z množiny povolených (*permitted*) a efektívnych (*effective*) oprávnení. Ak teda aj privilegovaný proces spustí iný program (napr. v dôsledku zneužitia zraniteľnosti útočníkom), príde o svoje oprávnenia. To podporuje skutočnosť, že subsystém *capabilities* slúži na prideľovanie oprávnení konkrétnym programom. Takto pridelené práva sa nededia na iné programy. Zavedenie množiny okolitých (*ambient*) oprávnení umožnilo procesu určiť, ktoré oprávnenia sa majú zachovať.

4.2.3. Oprávnenia subsystému *capabilities* a root

Ako sme spomenuli vyššie, v súčasných jadrách OS Linux sa na rozhodovanie o povolení alebo

zamietnutí vykonania privilegovanej operácie využíva subsystém *capabilities*. Aby bolo možné systém používať aj v tradičnom režime, teda tak, že privilegované operácie môže vykonať proces s efektívnym UID = 0 (resp. FSUID = 0 pri súborových operáciách), sú v jadre systému implementované dve špeciálne opatrenia – pri spúšťaní programu a pri zmene UID.

Ak je pri spúšťaní programu (operácii *execve*) reálne alebo efektívne (po spustení programu) UID procesu 0 (root), tak sa množiny povolených (*permitted*) a zdediteľných (*inheritable*) oprávnení programu považujú za plné (obsahujúce všetky oprávnenia). Ak efektívne UID = 0, tak sa bit *effective* programu považuje za nastavený na 1, inak na 0. Toto opatrenie má vplyv v dvoch typických situáciách – keď program spúšťa root, a keď bežný používateľ spúšťa program vlastnený root-om a s nastaveným *set-UID* bitom.

Keď program spúšťa root, reálne UID = 0. Preto sa toto špeciálne opatrenie použije a povolené oprávnenia procesu sa v súlade s algoritmom uvedeným v predchádzajúcej časti nastavia nasledovne:

- P'(permitted) = P(inheritable) | P(bounding_set)

Inými slovami, proces získa ako povolené (*permitted*) všetky oprávnenia, ktoré má v ohraničujúcej množine (*bounding set*) alebo ktoré mal pôvodne nastavené ako zdediteľné (*inheritable*). Efektívne UID procesu bude v tomto prípade zvyčajne tiež 0, a teda všetky povolené (*permitted*) oprávnenia budú nastavené aj ako efektívne (*effective*). Za predpokladu, že z ohraničujúcej množiny (*bounding set*) neboli odstránené žiadne oprávnenia, proces tak získa všetky oprávnenia a bude môcť vykonávať všetky privilegované operácie rovnako, ako v tradičnom bezpečnostnom modeli. Efektívne UID však nebude 0 v prípade, že root spustí program vlastnený iným používateľom a s nastaveným *set-UID* bitom. V takom prípade proces síce všetky oprávnenia bude mať v množine povolených (*permitted*), ale množina efektívnych (*effective*) bude prázdna. Tento stav je opäť konzistentný s tradičným bezpečnostným modelom.

Keď bežný používateľ spúšťa program vlastnený root-om a s nastaveným *set-UID* bitom, tak efektívne UID procesu bude 0, a teda špeciálne opatrenie pri spúšťaní programu sa taktiež aplikuje. Výsledkom bude nastavenie oprávnení:

- P'(permitted) = P(inheritable) | P(bounding_set)
- P'(effective) = P'(permitted)

Za predpokladu plnej ohraničujúcej množiny (*bounding set*) tak proces získa všetky oprávnenia aj ako povolené (*permitted*), aj ako efektívne (*effective*). Treba podotknúť, že toto opatrenie sa

neaplikuje v prípade, keď bežný používateľ spúšťa program, ktorý má explicitne priradené oprávnenia – v takom prípade sa použijú tie.

Ako sme už spomenuli aj skôr, tak odstránenie niektorého oprávnenia z ohraničujúcej množiny (*bounding set*) umožňuje zabrániť tomu, aby proces alebo jeho potomok neskôr získal toto oprávnenie. Ako vidíme, tak toto platí aj pri spúšťaní programov ako root, resp. programov, ktoré získavajú práva root-a pomocou *set-UID* mechanizmu.

Druhé špeciálne opatrenie sa aplikuje pri zmenách hodnôt UID atribútov procesu:

- Ak pred zmenou bolo aspoň jedno UID = 0 a po zmene sú všetky UID nenulové, všetky povolené (*permitted*) a efektívne (*effective*) oprávnenia procesu sa odstránia.
- Ak sa efektívne UID zmenilo z 0 na nenulovú hodnotu, všetky oprávnenia z množiny efektívnych (*effective*) oprávnení sa odstránia.
- Ak sa efektívne UID zmenilo z nenulovej hodnoty na 0, do množiny efektívnych (*effective*) oprávnení procesu sa skopíruje množina povolených (*permitted*) oprávnení.
- Ak sa FSUID zmenilo z 0 na nenulovú hodnotu, z množiny efektívnych (*effective*) oprávnení sa vymažú oprávnenie súvisiace so súborovými operáciami.
- Ak sa FSUID zmenilo z nenulovej hodnoty na 0, do množiny efektívnych (*effective*) oprávnení sa pridajú tie oprávnenia súvisiace so súborovými operáciami, ktoré sú v množine povolených (*permitted*) oprávnení.

Toto opatrenie má za cieľ dosiahnuť rovnaký efekt, ako v tradičnom bezpečnostnom modeli, keď efektívne oprávnenia záviseli od hodnoty efektívneho UID. Zároveň zabezpečuje nevratnosť oprávnení v prípade, že proces už nemá žiadne UID = 0, a teda nemá možnosť zmeniť efektívne UID na 0.

Dôsledkom týchto dvoch špeciálnych opatrení je, že systém, kde žiadny proces explicitne nemanipuluje s množinami oprávnení subsystému *capabilities*, a kde žiadny program nemá priradené oprávnenia, bude pracovať rovnako ako by pracoval pri použití tradičného bezpečnostného modelu bez subsystému *capabilities*. Priradením oprávnení programu docielime stav, kedy proces vykonávajúci tento program bude môcť vykonávať určené privilegované operácie aj bez toho, aby musel byť spustený v mene root-a. Subsystém *capabilities* však umožňuje aj zakázať aplikovanie týchto špeciálnych opatrení (prípadne iba niektorých ich častí). To, či sa uvedené špeciálne opatrenia aplikujú, je určené bitmi nazývanými *secure bits*:

- SECBIT_NOROOT
  - nastavením sa zakáže aplikácia špeciálneho opatrenia pri spúšťaní programu,
- SECBIT_NO_SETUID_FIXUP
  - nastavením sa zakáže úprava množín oprávnení pri zmene UID atribútov procesu,
- SECBIT_KEEP_CAPS
  - nastavením sa zakáže odstránenie všetkých oprávnení z množiny povolených (*permitted*) oprávnení pri zmene UID zo stavu „nejaká 0“ na „žiadna 0“ (súčasť špeciálneho opatrenia pri zmene UID),
- SECBIT_NO_CAP_AMBIENT_RAISE
  - nastavením sa zakáže procesu ďalšie pridávanie oprávnení do množiny okolitých (*ambient*) oprávnení.

Na zmenu týchto bitov slúži systémové volanie *prctl* a proces musí mať oprávnenie CAP_SETPCAP. Nastavenie týchto bitov sa dedí z procesu na jeho potomkov a bit SECBIT_KEEP_CAPS sa automaticky nuluje pri spustení programu (pri operácii *execve*). Stav týchto bitov je možné aj uzamknúť, čiže zabrániť akejkoľvek následnej zmene. Stav uzamknutia sa taktiež dedí z procesu na jeho potomkov, a toto uzamknutie je nevratné. Na uzamknutie slúžia ďalšie bity nastaviteľné volaním *prctl*:

- SECBIT_NOROOT_LOCKED,
- SECBIT_NO_SETUID_FIXUP_LOCKED,
- SECBIT_KEEP_CAPS_LOCKED,
- SECBIT_NO_CAP_AMBIENT_RAISE_LOCKED.

Štandardne sú všetky *secure bits* vrátane zámkov nulové (nenastavené), a teda špeciálne opatrenia pre kompatibilitu s tradičným bezpečnostným modelom sa aplikujú na všetky procesy. Ak však oprávnený proces (s CAP_SETPCAP) nastaví bity SECBIT_NOROOT a SECBIT_NO_SETUID_FIXUP a všetky tri zámky, a následne spustí nový program, zabezpečí, že jediným spôsobom, ako jeho potomkovia môžu získať nejaké oprávnenia, zostane len spustenie programu, ktorý má explicitne priradené nejaké oprávnenia. V rámci tohto podstromu procesov nebude mať identita root-a žiadny špeciálny význam. Netreba však zabudnúť na skutočnosť, že používateľ root je štandardne vlastníkom mnohých dôležitých súborov, čo mu umožňuje s nimi

manipulovať.

4.3. Zobrazenie a nastavenie oprávnení programu

Oprávnenia subsystému *capabilities* priradené programu sa pre účely zobrazovania alebo nastavovania popisujú textovým reťazcom, ktorý definuje postupnosť operácií s množinami oprávnení. Z historických dôvodov sa na tento účel aj *effective* bit považuje za množinu oprávnení, pričom táto musí byť buď prázdna (tento stav zodpovedá bitu s hodnotou 0) alebo musí obsahovať minimálne všetky oprávnenie, ktoré sa nachádzajú v množine povolených (*permitted*) alebo zdediteľných (*inheritable*) oprávnení programu (tento stav zodpovedá bitu s hodnotou 1).

Textový popis operácií sa skladá z postupnosti medzerami oddelených inštrukcií tvaru:

```
oprávneniaOperátorMnožiny[OperátorMnožiny...]
```

- *oprávnenia* je čiarkami oddelený zoznam oprávnení, príp. *all* (všetky oprávnenia),
- *Operátor* je
  - = odstrániť uvedené oprávnenia zo všetkých troch množín a pridať ich do uvedených množín,
  - + pridať uvedené oprávnenia do uvedených množín,
  - - odstrániť uvedené oprávnenia z uvedených množín,
- *Množiny* je zoznam množín, na ktoré sa má inštrukcia aplikovať – 0 až 3 znaky
  - **p** – povolené (*permitted*) oprávnenia,
  - **i** – zdediteľné (*inheritable*) oprávnenia,
  - **e** – efektívne (*effective*) oprávnenia.

Ak je v prvej inštrukcii operátor =, nemusí byť uvedený zoznam oprávnení – v takom prípade sa považuje za *all*.

Význam takéhoto popisu si môžeme priblížiť na príkladoch:

```
cap_net_raw,cap_net_admin=pe

cap_net_raw+p+e cap_net_admin+p+e

cap_net_raw,cap_net_admin=p cap_net_raw,cap_net_admin+e
```

Všetky tri vyššie uvedené popisy dosiahnu rovnaký efekt – množina povolených (*permitted*)

oprávnení bude obsahovať CAP_NET_RAW a CAP_NET_ADMIN, množina zdediteľných (*inheritable*) oprávnení bude prázdna a bit *effective* bude nastavený na 1.

Popisy

```
cap_net_raw+pe cap_net_admin+ie
```

```
cap_net_raw+p cap_net_admin+i all+e
```

tiež dosiahnu stav, kedy množina povolených (*permitted*) oprávnení obsahuje CAP_NET_RAW, množina zdediteľných (*inheritable*) oprávnení obsahuje CAP_NET_ADMIN a bit *effective* je nastavený na 1.

Pri používaní operátora = si treba dať pozor na to, že najprv uvedené oprávnenia odstráni zo všetkých množín. Napr.:

```
cap_net_raw=p all=e
```

spôsobí, že množina povolených (*permitted*) oprávnení bude prázdna, lebo operátor = z nej všetky oprávnenia odstráni.

Na nastavenie oprávnení programu slúži príkaz *setcap*:

```
setcap 'popis' súbor
```

Keďže popis musí byť v jednom argumente, tak, ak obsahuje medzery, je potrebné ho uzavrieť do úvodzoviek. Pri nastavovaní oprávnení je potrebné dať pozor na to, že tento príkaz nevie pridať nové oprávnenia k existujúcim – vždy existujúce nahradí oprávneniami určenými uvedeným popisom.

Ak chceme oprávnenia programu odstrániť, môžeme použíť príkaz *setcap* nasledovne:

```
setcap -r súbor
```

Na nastavenie alebo odstránenie oprávnení programu je potrebné mať oprávnenie CAP_SETFCAP.

Na zobrazenie oprávnení slúži príkaz *getcap*:

```
getcap [-r] súbor
```

Príkaz vypíše oprávnenia uvedeného programu v podobe vyššie popísaného textového popisu. Prepínač *-r* umožňuje rekurzívne vypísať oprávnenia všetkých súborov v podstrome.

Použitie subsystému *capabilities* si môžeme demonštrovať na nasledujúcom príklade. Použijeme klasický program *ping* na testovanie dostupnosti uzla siete pomocou protokolu ICMP. Na odosielanie a prijímanie ICMP paketov je potrebné oprávnenie CAP_NET_RAW. Na systémoch,

ktoré nevyužívajú subsystém *capabilities* máva preto tento program štandardne nastavený *set-UID* bit a je vlastnený root-om. Keď spustíme kópiu programu *ping* bez *set-UID* bitu, nebude fungovať, pretože mu bude chýbať uvedené oprávnenie:

```
$ ls -l ping
-rwxr-xr-x 1 root root 31104 Oct  7 13:43 ping
$ ./ping 127.0.0.1
ping: icmp open socket: Operation not permitted
```

Teraz môžeme programu *ping* nastaviť oprávnenie CAP_NET_RAW:

```
# setcap CAP_NET_RAW+pe ping
# getcap ping
ping = cap_net_raw+ep
```

Keď ho bežný používateľ spustí teraz, program už fungovať bude:

```
$ ./ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.049 ms
^C
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.049/0.049/0.049/0.000 ms
```

4.4. Využitie subsystému capabilities

Ako sme videli na predchádzajúcom príklade, subsystém *capabilities* nám umožňuje jednoduchým spôsobom prideliť vybrané oprávnenia programom, čím sa môžeme vyhnúť potrebe spúšťať ich ako root, resp. používať *set-UID* mechanizmus na získanie práv root-a. Uveďme si ešte niekoľko typov programov, kde môže byť tento prístup úspešne aplikovaný:

- programy, ktoré potrebujú prístup k RAW paketom
  - potrebujú CAP_NET_RAW
    - ping, tcpdump, ...

- programy, ktoré počúvajú na TCP/UDP portoch < 1024
  - potrebujú CAP_NET_BIND_SERVICE
    - rôzne sieťové služby
- zálohovacie programy
  - potrebujú CAP_DAC_READ_SEARCH

4.5. Podpora subsystému capabilities v OS Linux

Na nastavovanie a zobrazovanie oprávnení programov pomocou príkazov *setcap* a *getcap* je potrebné mať nainštalovaný balík *libcap2-bin*. Ďalší užitočný balík je *libcap-ng-utils*, ktorý obsahuje príkazy *netcap*, *filecap*, *pscap*, *captest*, ktoré umožňujú jednoduchým spôsobom zobraziť, aké oprávnenia subsystému *capabilities* majú aktuálne jednotlivé procesy (*pscap*), procesy, ktoré používajú sieťovú komunikáciu (*netcap*) alebo zobraziť a prípadne nastaviť oprávnenia programom (*filecap*).

Oprávnenia programov sú ukladané, podobne ako ACL, v rozšírených atribútoch (*extended attributes*). Na ich používanie je preto potrebné, aby programy boli uložené v súborovom systéme, ktorý tieto atribúty podporuje.

Ďalším užitočným (a zvyčajne automaticky nainštalovaným pri inštalácii *libcap2-bin*) balíkom je *libpam-cap*. Tento obsahuje modul pre autentifikačný subsystém PAM, ktorý môže byť využitý na nastavenie zdieľateľných (*inheritable*) oprávnení pri prihlásení používateľa. Typicky sa používa pridaním riadku

```
auth    optional                        pam_cap.so
```

do súboru */etc/pam.d/common-auth*. Je konfigurovaný jednoduchým textovým súborom */etc/security/capability.conf*, ktorý obsahuje riadky v tvare:

```
zoznam_oprávnení     meno_používateľa
```

- *zoznam_oprávnení* obsahuje čiarkami oddelený zoznam oprávnení, ktoré majú byť používateľovi nastavené do množiny zdediteľných (*inheritable*) oprávnení, *none* znamená, že množina má byť explicitne vyprázdnená,
- *meno_používateľa* obsahuje meno používateľa alebo * (akýkoľvek používateľ).

Použije sa prvý riadok, ktorý zodpovedá prihlasujúcemu sa používateľovi (teda riadok s * by mal byť uvedený ako posledný).

Tento modul môžeme využiť napr. na to, aby sme programom mohli prideliť oprávnenia v závislosti od toho, či ich spustil vybraný používateľ alebo iný používateľ. Ak programu priradíme oprávnenia do množiny zdediteľných (*inheritable*) oprávnení namiesto do množiny povolených (*permitted*) oprávnení, získa tieto oprávnenia len vtedy, ak ich spúšťajúci proces bude mať vo svojej množine zdediteľných. A keďže množina zdediteľných oprávnení sa medzi procesmi dedí, tak tento predpoklad môžeme splniť napríklad pomocou tohto PAM modulu. Môžeme si to demonštrovať na príklade, keď chceme programu *tcpdump* dať potrebné oprávnenia na zachytávanie sieťovej komunikácie, ak ho spustí používateľ *jerry*.

Najprv nastavíme zdediteľné oprávnenia programu *tcpdump*:

```
# ls -la /usr/sbin/tcpdump
-rwxr-xr-x 1 root root 822744 Dec  2 12:17 /usr/sbin/tcpdump
# setcap cap_net_raw+ie /usr/sbin/tcpdump
# getcap /usr/sbin/tcpd
tcpd       tcpdchk    tcpdmatch  tcpdump
# getcap /usr/sbin/tcpdump
/usr/sbin/tcpdump = cap_net_raw+ei
```

Do */etc/security/capability.conf* pridáme riadok:

```
cap_net_raw     jerry
```

Po prihlásení sa používateľa *jerry* tento bude mať možnosť využiť oprávnenie CAP_NET_RAW prostredníctvom programov, ktoré toto oprávnenie majú nastavené ako zdediteľné (napr. náš *tcpdump*):

```
$ /usr/sbin/tcpdump -nlp -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
02:00:43.147330 IP 158.195.87.39.22 > 158.195.87.244.53252: Flags [P.], seq
1554557296:1554557488, ack 4108854434, win 1541, options [nop,nop,TS val
1433547080 ecr 624627715], length 192
02:00:43.147559 IP 158.195.87.39.22 > 158.195.87.244.53252: Flags [P.], seq
192:416, ack 1, win 1541, options [nop,nop,TS val 1433547080 ecr
624627715], length 224
...
```

Výhodou tohto riešenia oproti tomu, keby sme programu nastavili oprávnenia ako povolené (*permitted*) a pomocou riadenia prístupu povolili jeho spúšťanie len vybraným používateľom, je, že takto môžu program spustiť aj iní používatelia, ale bez toho, aby získal oprávnenia. To môže byť užitočné pre programy, ktoré majú aj bez oprávnení zmysel (napr. *tcpdump* na analýzu uloženej zachytenej sieťovej komunikácie, alebo *tar* na archiváciu vlastných údajov, pričom s oprávnením CAP_DAC_READ_SEARCH by mohol byť použitý na zálohovanie všetkých údajov).

5. Subsystém SELinux

Subsystém SELinux je bezpečnostný modul (LSM – *Linux Security Module*) jadra OS Linux, ktorý umožňuje aplikovať ďalšie obmedzenia na vykonávanie mnohých bezpečnostne relevantných operácií v systéme. Pokrýva ako bežné operácie (napr. prístup k súborom), tak aj privilegované operácie (inak kontrolované subsystémom *capabilities*). Ako uvidíme neskôr, rozlíšenie operácií je pri tomto subsystéme výrazne jemnejšie ako v tradičnom bezpečnostnom modeli. Navyše, napr. prístup k súborom sa nekontroluje len pri otváraní súboru, ale aj pri každom čítaní či zápise.

Subsystém SELinux predstavuje prostriedok povinného riadenia prístupu (MAC – *Mandatory Access Control*), nakoľko nie je pod kontrolou bežného používateľa – vlastníka objektu. Rozhodovanie o tom, či SELinux operáciu povolí alebo nie, je riadené flexibilnou bezpečnostnou politikou na základe niekoľkých bezpečnostných atribútov subjektov a objektov. Táto politika taktiež určuje, do akej miery môžu byť tieto atribúty ovplyvnené rôznymi používateľmi.

Aby bola operácia povolená, musí byť povolená zároveň tradičnými subsystémami, a zároveň subsystémom SELinux, tzn. SELinux nepredstavuje alternatívu k tradičnému riadeniu prístupu, ani neumožňuje nové výnimky z tradičného modelu. SELinux tiež zasahuje do povoľovania privilegovaných operácií kontrolovaných subsystémom *capabilities*. Aj v tomto prípade však predstavuje len ďalšie obmedzenia, nie výnimky. Teda privilegovaná operácia bude povolená len vtedy, ak ju povolí subsystém *capabilities* a zároveň SELinux.

5.1. Architektúra subsystému SELinux

Subsystém SELinux pozostáva z niekoľkých dôležitých komponentov:

- *Security Server* (bezpečnostný server),
- *Security Policy* (bezpečnostná politika),
- *Access Vector Cache* (cache povolených prístupov),
- *Object Manager* (správca objektov).

Object Manager (správca objektov) je komponent, ktorý je zodpovedný za vykonávanie operácií s určitými objektami. Každá operácia, ktorá má byť kontrolovateľná subsystémom SELinux, musí byť vykonateľná výlučne prostredníctvom služieb tohto komponentu. Ten vďaka tomu môže vynútiť určitú bezpečnostnú politiku – rozhodnutie o tom, či operácia môže alebo nemôže byť vykonaná.

Príkladom správcu objektov je jadro (kernel) OS Linux. To zabezpečuje napr. prístup k súborom, k

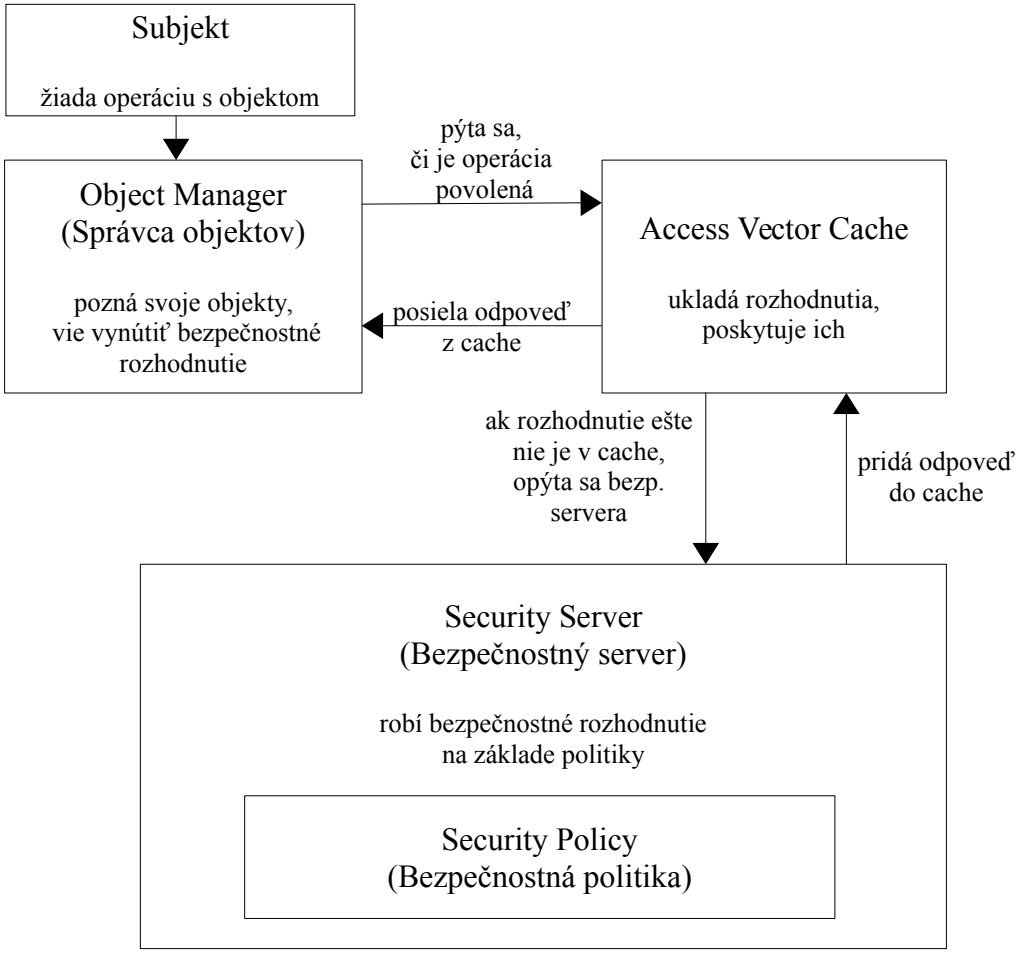
systémovým parametrom, k sieti, k iným procesom, ... Avšak, správcom objektov môže byť aj nejaký aplikačný subsystém – napr. databázový server, X-Windows server, a pod. Aj takíto spravcovia objektov môžu využívať zvyšok subsystému SELinux na riadenie prístupu k svojim objektom. V prípade databázového systému môžu byť objektami napr. databázy, tabuľky, stĺpce tabuliek, a pod. To umožňuje podstatne jemnejšie riadenie prístupu ako len na úrovni jadra OS.

Security Policy (bezpečnostná politika) je súbor pravidiel, ktoré určujú či daný subjekt môže vykonať určitú operáciu s daným objektom. Ďalej obsahuje pravidlá, ktoré určujú, ako sa majú inicializovať alebo meniť atribúty subjektov a objektov pri rôznych operáciách.

Security Server (bezpečnostný server) je komponent, ktorý používa bezpečnostnú politiku na výpočet rozhodnutí o povolení či zamietnutí operácie, na výpočet hodnôt atribútov pri inicializácii subjektov a objektov, či pri operáciách, ktoré spôsobujú zmenu atribútov. Keďže bezpečnostná politika môže byť pomerne zložitá, aj jej spracovanie a vyhodnotenie je pomerne časovo náročná operácia. A keďže správca objektov potrebuje rozhodnutia bezpečnostného servera pomerne často, neobracia sa priamo na bezpečnostný server, ale využíva cache povolených prístupov.

Access Vector Cache (cache povolených prístupov) obsahuje informácie o povolených operáciách pre dvojicu (subjekt, objekt), ktorú už bezpečnostný server na základe politiky určil. To umožňuje opakované použitie tejto informácie bez toho, aby ju bolo potrebné znova určovať na základe bezpečnostnej politiky. Účinnosť tohto prístupu vychádza zo skutočnosti, že bezpečnostná politika sa často nemení. V prípade, že sa politika zmení, obsah cache sa, samozrejme, zneplatní a informácia o povolených operáciách sa znovu určí a uloží do cache, keď bude najbližšie potrebná.

Vzťah medzi komponentami subsystému SELinux je zobrazený na nasledujúcom obrázku.



5.2. Bezpečnostné mechanizmy SELinux-u

SELinux Implementuje niekoľko bezpečnostných mechanizmov:

- *Role-based Access Control* (RBAC)
  - riadenie prístupu založené na rolách,
- variant modelu DTE – *Domain and Type Enforcement*,
  - riadenie prístupu založené na určení povolených operácií medzi tzv. *doménou* subjektu a *typom* objektu,
- voliteľne MLS – *Multi-Level Security*
  - riadenie prístupu k hierarchicky klasifikovaným údajom – umožňuje implementovať modely ako Bell – La Padula alebo Biba (pravidlá sú konfigurovateľné).

5.2.1. SELinux DTE

V modeli DTE má každý objekt priradený bezpečnostný atribút nazývaný *typ*, a každý subjekt má priradený bezpečnostný atribút nazývaný *doména*. Bezpečnostná politka DTE určuje, aké operácie môže subjekt v danej doméne vykonať s objektom daného typu. Taktiež určuje, aký má byť typ nového objektu pri jeho vzniku, či v akej doméne má proces bežať po spustení programu v závislosti od pôvodnej domény procesu a typu súboru, z ktorého je program spustený.

DTE je veľmi flexibilný bezpečnostný mechanizmus, pomocou ktorého je možné implementovať veľmi širokú škálu bezpečnostných politík. Umožňuje, napríklad, veľmi podrobne určiť, ku ktorým súborom môže ktorý program pristupovať. Vďaka tomu je možné významne znížiť potenciálny dopad zneužitia nejakej zraniteľnosti v programe. Môžeme si to priblížiť na príkladoch.

Ako prvý príklad uvažujme jednoduchý web server. Jeho úlohou je prostredníctvom siete poskytovať prístup k niektorým dokumentom. Musí preto môcť:

- čítať sprístupňované dokumenty,
- čítať systémové knižnice a niektoré konfiguračné súbory,
- pridávať záznamy do logu,
- komunikovať prostredníctvom TCP protokolu na príslušnom porte.

Pomocou DTE mechanizmu v SELinux-e môžeme definovať, že proces v doméne pre web server môže robiť len tieto uvedené operácie, ale nemôže napr. manipulovať s inými súbormi. Ak potom aj dôjde k zneužitiu nejakej zraniteľnosti v programe web servera a útočník bude schopný v rámci procesu web servera vykonávať vlastný kód, získa prístup len k tým operáciám, ktoré má doména web servera povolené.

Ako druhý príklad uvažujme používanie asymetrickej kryptografie na autentifikáciu používateľa voči vzdialenému systému (napr. pomocou *ssh*). V bežnom prípade je súkromný kľúč používateľa uložený v súbore v podadresári domovského adresára. Tento súbor je chránený pomocou klasického riadenia prístupu tak, že k nemu má prístup len jeho vlastník. Ak však používateľ spustí nejakú škodlivú aplikáciu alebo sa inak neškodná ale zraniteľná aplikácia stane škodlivou v dôsledku zneužitia jej zraniteľnosti útočníkom, táto môže prečítať používateľov súkromný kľúč a sprístupniť ho útočníkovi. Ak je tento súbor chránený šifrovaním, tak útočník ešte bude musieť uhádnuť alebo získať heslo, ale už získanie súboru s kľúčom je narušením prvej línie ochrany. Pomocou DTE v SELinux-e môžeme definovať doménu pre programy, ktoré môžu pristupovať k súkromnému kľúču, definovať typ súboru pre súkromné kľúče, a určiť, že pracovať so súkromným kľúčom

môžu len procesy v príslušnej doméne. Tým zabránime, aby iná aplikácia mohla získať prístup k citlivému súboru.

Ako vidíme z uvedených príkladov, DTE nám umožňuje definovať podrobné prístupové práva pre konkrétne programy. Tým rieši typický problém klasického voliteľného riadenia prístupu – zneužívanie prístupových práv používateľa škodlivými aplikáciami.

SELinux implementuje variant DTE, v ktorom formálne nerozlišuje medzi doménami a typmi, takže môžeme hovoriť aj o type subjektu. Subjekty (procesy) a objekty (napr. objekty súborového systému, procesy ako cieľ operácie, sieťové pakety, ...) majú priradený *typ*. Pre lepšiu zrozumiteľnosť však budeme v prípade subjektov ich typ naďalej nazývať aj *doménou*. Politika určuje povolené operácie na základe typu (domény) subjektu, typu objektu a *triedy* (*class*) objektu. Trieda objektu určuje, o objekt akého druhu sa jedná, napr. súbor, adresár, socket, proces, ... Trieda objektu určuje množinu kontrolovaných operácií, resp. množinu práv, ktoré môžu byť subjektu udelené vo vzťahu k objektu. Vďaka tomu, že sa pri rozhodovaní zohľadňuje aj trieda, môžeme použiť rovnaký typ pre rôzne druhy objektov. Typickým príkladom môže byť použitie rovnakého typu pre adresáre aj súbory slúžiace spoločnému účelu. Vďaka triede potom vieme rozlišovať medzi právami na súbory a právami na adresáre. Niektoré objekty (napr. socket-y), ktoré nemôžu existovať bez spojitosti s procesom, automaticky dedia typ procesu, ktorý ich vytvoril. Opäť vďaka používaniu triedy je možné rozlišovať medzi právami na proces a právami na takéto objekty, ktoré vytvoril.

5.2.2. SELinux RBAC a používatelia

Riadenie prístupu na základe rol vo všeobecnosti znamená, sa práva neprideľujú priamo používateľom, ale rolám, v ktorých môžu používatelia vystupovať. Používateľom sa následne prideľujú roly, čím získavajú príslušné práva.

V SElinux-e sa RBAC používa na prideľovanie oprávnení na vykonávanie programu v doméne. Každý subjekt (proces) má priradeného používateľa, v mene ktorého koná, rolu, v ktorej koná a doménu. To, aké hodnoty môže trojica atribútov (používateľ, rola, doména) procesu nadobúdať, je však politikou obmedzené. Každý používateľ má pridelené roly, v ktorých môže vystupovať. Každá rola má pridelené domény, v ktorých môžu bežať procesy konajúce v tejto role. Aby mohol mať proces priradenú trojicu (používateľ, rola, doména), musí platiť, že uvedený používateľ je oprávnený vystupovať v uvedenej role, a že uvedená rola je má pridelenú uvedenú doménu. Týmto spôsobom je možné obmedziť použitie domény len na niektoré roly a obmedziť, v ktorých rolách môže konkrétny používateľ vystupovať.

Napríklad, je pomerne bežné obmedziť použitie domén, ktoré majú prístup ku konfigurácii systému, na administrátorskú rolu. Tým je zabezpečené, že používateľ, ktorý nie je oprávnený vystupovať v administrátorskej role, nebude môcť spustiť proces v doméne, ktorá má prístup ku konfigurácii systému, a teda nebude môcť do tejto konfigurácie zasiahnuť.

Používateľ môže byť oprávnený vystupovať vo viacerých rolách, ale každý jeho proces v danom čase koná práve v jednej role. Zmena roly je operácia, ktorú SELinux umožňuje kontrolovať. Typicky je možná len použitím programu bežiaceho v špeciálnej doméne, ktorý si vyžaduje interaktívnu autentifikáciu používateľa. Ak používateľ oprávnený vystupovať v administrátorskej role napr. štandardne po prihlásení vystupuje v neprivilegovanej role, nie je možné, aby nejaký škodlivý proces bez vedomia používateľa zmenil rolu na administrátorskú a spustil nejaký program v privilegovanej doméne. Navyše, politika presne určuje z akej roly do akej roly je povolené prejsť. Vďaka tomu, napríklad, bežní používatelia nijakým spôsobom nemôžu realizovať zmenu roly na administrátorskú.

Vzhľadom na to, že Linux-ová identita používateľa (hodnoty UID atribútov) sa v Linux-e často menia (napr. pomocou *set-UID* mechanizmu), SELinux nepoužíva Linux-ovú identitu používateľa. Namiesto toho používa vlastné identifikátory používateľov. Pri prihlásení používateľa je mu pridelená SELinux-ová identita používateľa podľa mapovania, ktoré jednotlivým Linux-ovým používateľským menám priraďuje SELinux-ové identifikátory používateľov. SELinux-ová identita používateľa následne zostáva spravidla zachovaná počas celej doby práce používateľa bez ohľadu na to, ako sa prípadne priebežne mení jeho Linux-ová identita.

Vzhľadom na to, akým spôsobom je realizované priraďovanie SELinux-ovej identity používateľom, sú možné viaceré scenáre. Je možné každému Linux-ovému používateľovi vytvoriť individuálnu SELinux-ovú identitu, ale je tiež možné použiť rovnakú SELinux-ovú identitu pre viacero rôznych Linux-ových používateľov. V druhom prípade nebude SELinux medzi týmito používateľmi vedieť rozlišovať, čo však nemusí predstavovať problém. Netreba zabúdať, že popri SELinux-e stále funguje aj klasické riadenie prístupu, ktoré medzi týmito používateľmi rozlišuje.

5.2.3. SELinux MLS

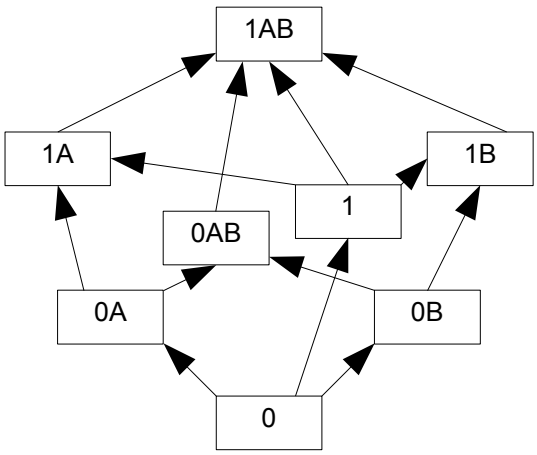
Voliteľnou súčasťou SELinux-u je aj podpora MLS. Princíp MLS je známy napríklad zo oblasti utajovaných skutočností. Môžeme si ho predstaviť na príklade modelu Bell – La Padula. Objekty sú klasifikované stupňom utajenia a množinou kategórií, ktorých sa týkajú. Subjekty majú pridelené oprávnenia na určitý stupeň utajenia a množinu kategórií. Dvojicu (stupeň utajenia, množina kategórií) nazvime bezpečnostnou značkou. Stupne utajenia sú usporiadané od najnižšieho

(najmenej tajné) po najvyšší (najviac tajné). Bezpečnostné značky tvoria čiastočné usporiadanie definované vzťahom:

$$(s_1, C_1) \geq (s_2, C_2) \Leftrightarrow s_1 \geq s_2 \wedge C_1 \supseteq C_2$$

kde s_i sú stupne utajenia a C_i sú množiny kategórií.

Vzájomné vzťahy medzi bezpečnostnými značkami môžeme demonštrovať na jednoduchom príklade. Predpokladajme, že máme len 2 stupne utajenia (0 a 1) a dve kategórie (A a B). Nasledujúci obrázok ilustruje vzťah čiastočného usporiadanie bezpečnostných značiek, ktoré môžeme vytvoriť. Ak existuje cesta po orientovaných hranách (šípkach) od značky *x* ku značke *y*, tak $y \geq x$, čize po orientovaných hranách je možné prejsť z „nižšej“ bezpečnostnej značky k „vyššej“.



Vidíme, že najnižšou bezpečnostnou značkou je tá so stupňom 0 a prázdnou množinou kategórií a najvyššou tá so stupňom 2 a množinou kategórií {A, B}. Ďalej vidíme, že napríklad $(0, \{A\}) \leq (1, \{A,B\})$, $(1, \{B\}) \leq (1, \{A,B\})$, ale že napríklad $(1, \{A\})$ a $(1, \{B\})$ sú navzájom neporovnateľné.

Cieľom modelu Bell – La Padula je zabezpečiť ochranu dôvernosti, teda zaistiť, aby subjekt nemohol prečítať informáciu z objektu s bezpečnostnou značkou, ktorá nie je nižšia alebo rovnaká ako bezpečnostná značka (vyjadrujúca oprávnenie) subjektu. Táto vlastnosť má zostať zachovaná aj v prípade, že na úniku informácie sa rozhodnú spolupracovať viaceré subjekty, pričom niektoré môžu byť aj oprávnené informáciu zo zdrojového objektu prečítať. Teda je potrebné zabezpečiť aj to, aby subjekt oprávnený čítať z nejakého objektu nemohol takto získané informácie sprístupniť subjektu, ktorý na prečítanie tohto objektu oprávnený nie je. Na naplnenie týchto požiadaviek definuje model Bell – La Padula dve jednoduché pravidlá. Nech subjekt má bezpečnostnú značku S a objekt bezpečnostnú značku O.

Potom

- subjekt môže **čítať** z objektu vtedy a len vtedy, keď $S \geq O$ (*no read up*),
- subjekt môže **zapisovať** do objektu vtedy a len vtedy, keď $S \leq O$ (*no write down*).

Existujú aj verzie tohto modelu, kde sa druhom pravidle (pre zápis) vyžaduje dokonca rovnosť bezpečnostných značiek subjektu a objektu. Tieto pravidlá umožňujú, aby informácie medzi objektami a subjektami s rôznymi bezpečnostnými značkami tiekli len smerom zdola nahor, teda z objektu / subjektu s nižšou bezpečnostnou značkou do objektu / subjektu s vyššou bezpečnostnou značkou. Prenos smerom nadol, ani prenos medzi subjektom a objektom s neporovnateľnou bezpečnostnou značkou je znemožnený. Aby bolo možné preniesť informácie aj zakázaným smerom, model definuje špeciálnu kategóriu tzv. *dôveryhodných* subjektov (*trusted subjects*), pre ktoré neplatí pravidlo pre zápis. Inými slovami, dôveryhodné subjektu môžu „odtajňovať" informácie, a teda ich zapísať aj do objektu s nižšou (príp. neporovnateľnou) bezpečnostnou značkou.

Iným známym modelom MLS je model Biba, ktorý slúži na ochranu integrity údajov. Je založený na rovnakom princípe ako Bell – La Padula, ale porovnávacie operátory v pravidlách sú presne opačné ako v Bell – La Padula modeli. Namiesto stupňa utajenia sa používa stupeň dôveryhodnosti, a model sleduje znemožnenie toho, aby menej dôveryhodná informácia mohla ovplyvniť dôveryhodnejšiu.

Podpora pre MLS v SELinux-e umožňuje pomerne flexibilne implementovať modely založené na rovnakom princípe ako Bell – La Padula či Biba. Každému objektu a subjektu priraďuje dve bezpečnostné značky (dolnú a hornú) a umožňuje, aby politika definovala obmedzenia, ktoré musia byť splnené pre vykonanie operácie medzi subjektom a objektom. Tieto obmedzenia môžu porovnávať bezpečnostné značky, ako aj DTE domény a typy, čo umožňuje napríklad realizovať aj výnimky pre dôveryhodné subjekty na základe domény subjektu. Keďže obmedzenia sú pomerne voľne konfigurovateľné, je možné implementovať rôzne varianty Bell – La Padula modelu, rôzne varianty Biba modelu, alebo iných podobných modelov.

Idea MLS síce pôvodne pochádza z oblasti utajovaných skutočností, no je využiteľná aj v civilnom prostredí. V SELinux-e sa často používa variant MLS nazývaný aj MCS (*Multi-Category Security*), ktorý nevyužíva prvú zložku bezpečnostných značiek (úroveň) ale len druhú (množina kategórií). Formálne je úroveň nutná súčasť bezpečnostnej značky, ale je možné definovať aj len jednu úroveň, čím efektívne stráca význam – bezpečnostné značky sa potom porovnávajú len na základe kategórií. Využitiu MCS v praxi sa budeme venovať neskôr.

5.2.4. Bezpečnostný kontext

Ako sme videli v predchádzajúcich častiach, jednotlivé bezpečnostné mechanizmy SELinux-u potrebujú, aby subjekty a objekty mali priradených niekoľko nových atribútov. Tieto atribúty v SELinux-e spoločne tvoria tzv. *bezpečnostný kontext* (*security context*). Každý subjekt a objekt má teda priradený bezpečnostný kontext. Pri komunikácii s používateľom sa bezpečnostný kontext reprezentuje v textovej forme v nasledujúcom tvare:

```
user:role:type[:mls_range]
```

Jednotlivé atribúty sú oddelené dvojbodkami, prvé tri musia byť uvedené vždy, štvrtý je uvedený v prípade, ak daná verzia SELinux-ovej politiky má zapnutú podporu MLS. Význam jednotlivých položiek je nasledovný:

- *user* – meno SELinux-ového používateľa priradeného subjektu / objektu,
- *role* – aktuálna rola subjektu / objektu,
- *type* – typ (doména) subjektu / objektu,
- *mls_range* – rozsah MLS značiek.

Keďže rola je atribút, ktorý má význam len pre subjekty, SELinux používa špeciálnu preddefinovanú rolu *object_r* pre objekty.

Rozsah MLS značiek pozostáva z dvojice značiek – dolnej (*low*) a hornej (*high*). MLS bezpečnostná značka pozostáva z dvoch častí – úrovne a množiny kategórií. Úrovne aj jednotlivé kategórie majú svoje identifikátory definované v politike. Ak je množina kategórií prázdna, značka sa zapisuje len ako identifikátor úrovne (napr. *s0*). Ak množina kategórií nie je prázdna, oddeľuje sa úroveň od zoznamu kategórií dvojbodkou a jednotlivé kategórie sa oddeľujú čiarkami (napr. *s0:c1,c3*). Kategórie majú určené svoje prirodzené poradie dané poradím ich definovania v politike, a ak chceme zapísať množinu, ktorá obsahuje niekoľko prirodzene bezprostredne po sebe nasledujúcich kategórií, môžeme zápis skrátiť uvedením prvej a poslednej takej kategórie, pričom ich oddelíme bodkou (napr. *s0:c1.c3* má rovnaký význam ako *s0:c1,c2,c3*, ak kategórie c1, c2 a c3 sú v politike definované takto za sebou). Ak sú dolná a horná značka rovnaké, v *mls_range* sa zapisuje len jedna, ak sú rôzne, oddeľujú sa pomlčkou. Napríklad rozsah MLS značiek

```
s0-s0:c0.c10
```

znamená:

- dolná značka *s0*
    - úroveň *s0*, prázdna množina kategórií
- horná značka *s0:c0.c10*
    - úroveň *s0*, množina obsahujúca všetky kategórie *c0* až *c10*.

Rozsah MLS značiek

```
s0:c0-s1:c0,c4
```

znamená:

- dolná značka *s0:c0*
    - úroveň *s0*, kategória *c0*
- horná značka *s1:c0,c4*
    - úroveň *s1*, kategórie *c0* a *c4*.

Rozsah MLS značiek

```
s0:c1,c2
```

znamená:

- dolná aj horná značka *s0:c1,c2*
    - úroveň *s0*, kategórie *c1* a *c2*.

Príklad kompletného bezpečnostného kontextu môže vyzerať nasledovne:

```
user_u:user_r:user_t:s0-s0:c0.c10
```

a jeho význam je:

- používateľ *user_u*,
- rola *user_r*,
- typ (doména) *user_t*,
- rozsah MLS značiek:
    - dolná: *s0* s prázdnou množinou kategórií,
    - horná: *s0* s kategóriami *c0* až *c10*.

Bezpečnostný kontext objektov súborového systému môže byť určený niekoľkými spôsobmi v závislosti od typu súborového systému:

- uložený v rozšírených atribútoch (*extended attributes*)
  - tento spôsob sa používa pre súborové systémy, ktoré podporujú rozšírené atribúty,
- odvodený od vytvárajúceho procesu
  - tento spôsob sa používa pre špeciálne pseudosúborové systémy, ktoré v podobe súborov prezentujú rôzne objekty naviazané na procesy (napr. socket-y, rúry (*pipe*), a pod.),
- určený pravidlom politiky na základe vytvárajúceho procesu a typu súborového systému
  - tento spôsob sa používa pre špeciálne súborové systémy ako napr. *tmpfs, shm*, a pod.,
- určený politikou jednotne pre celý súborový systém
  - tento spôsob sa používa pre súborové systémy bez podpory rozšírených atrtibútov, ako napr. *vfat*.

Bezpečnostný kontext pre subjekty a iné objekty je zvyčajne zdedený od vytvárajúceho subjektu.

Bežné príkazy v dnešných distribúciách OS Linux, ktoré typicky zobrazujú rôzne atribúty súborov či procesov, vedia zobrazovať aj bezpečnostný kontext. Zvyčajne na to slúži prepínač *-Z*.

V prvom príklade vidíme zobrazenie bezpečnostného kontextu „používateľa“ – v skutočnosti ide o bezpečnostný kontext procesu, v rámci ktorého je vykonávaný program *id*, čo však zvyčajne zodpovedá bezpečnostnému kontextu, ktorý je priradený bežným programom spusteným týmto prihláseným používateľom:

```
admin@debian:~$ id -Z
staff_u:staff_r:staff_t:s0
```

V nasledujúcom príklade vidíme zobrazenie bezpečnostného kontextu bežiacich procesov pri použitím príkazu *ps*:

```
admin@debian:~$ ps axZ
LABEL                             PID TTY      STAT   TIME COMMAND
staff_u:staff_r:staff_t:s0       4993 pts/1    Ss     0:00 -bash
staff_u:staff_r:staff_t:s0       5158 pts/1    R+     0:00 ps axZ
```

V ďalšom príklade vidíme zobrazenie bezpečnostného kontextu súborov pri použití príkazu *ls*:

```
root@debian:~# ls -laZ /etc/passwd /etc/shadow /bin/bash /bin/cp
-rwxr-xr-x. 1 root root   system_u:object_r:shell_exec_t:s0 941252 Jan  1  2013 /bin/bash
-rwxr-xr-x. 1 root root   system_u:object_r:bin_t:s0        124804 Jan 26  2013 /bin/cp
-rw-r--r--. 1 root root   system_u:object_r:etc_t:s0          1110 Oct  8 18:22 /etc/passwd
-rw-r-----. 1 root shadow system_u:object_r:shadow_t:s0       1091 Oct  8 21:35 /etc/shadow
```

5.3. Bezpečnostná politika SELinux-u

Bezpečnostná politika SELinux-u podrobne určuje, aké rozhodnutia bezpečnostný server SELinux-u spraví. Bezpečnostná politika sa píše v textovom tvare vo vlastnom jazyku a následne sa transformuje do binárnej podoby. Binárna podoba bezpečnostnej politiky je potom nahraná do jadra OS Linux, kde sa ňou následne riadi bezpečnostný server pri svojej činnosti.

Bezpečnostná politika SELinux-u pozostáva z viacerých typov deklarácií:

- definícia typov a atribútov typov,
- povoľovacie a auditovacie pravidlá,
- pravidlá pre odvodzovanie typov,
- definícia rol, povolených domén pre roly a povoľovacích pravidiel pre prechod medzi rolami,
- definícia používateľov a povolených rol,
- definícia obmedzení,
- definícia MLS značiek,
- definícia MLS pravidiel,
- definícia bezpečnostných kontextov pre niektoré objekty (napr. sieťové rozhrania, porty, sieťové adresy, súborové systémy, …).

V nasledujúcich častiach si postupne rozoberieme syntax a význam vybraných deklarácií bezpečnostnej politiky.

5.3.1. Definícia typov a atribútov typov

Každý typ DTE politiky SELinux-u má svoj identifikátor. Bežná konvencia pri pomenovávaní typov v SELinux-e používa príponu *_t* na označenie, že identifikátor označuje typ. Každý typ môže mať priradených niekoľko tzv. *atribútov* typu. Atribúty slúžia na to, aby bolo možné v pravidlách vyjadriť „akýkoľvek typ, ktorý má priradený daný atribút“. Typické je napr. použitie atribútu, ktorým budú označené všetky typy používané ako domény, teda typy pre subjekty. To potom umožňuje napísať jedno pravidlo, ktorým sa určité právo pridelí všetkým subjektom bez toho, aby sme museli explicitne uviesť všetky existujúce domény.

Typ sa v politike definuje pomocou deklarácie v jednom z nasledujúcich tvarov:

```
type id_t;
type id_t, attr1, attr2;
```

Obe deklarácie definujú typ *id_t*, druhá mu navyše hneď aj priradí atribúty *attr1* a *attr2*. Atribúty musia vopred tiež definované, a to deklaráciou v nasledujúcom tvare:

```
attribute attr1;
```

Táto deklarácia definuje atribút *attr1*. Ak potrebujeme typu dodatočne (teda nie hneď pri jeho definovaní) priradiť nejaký atribút, slúži na to deklarácia v tvare:

```
typeattribute id_t attr1, attr2 ;
```

Táto priradí existujúcemu typu *id_t* atribúty *attr1* a *attr2*.

5.3.2. Povoľovacie a auditovacie pravidlá

Každá operácia, ktorá nie je explicitne povolená nejakým povoľovacím pravidlom v politike je SELinux-om štandardne zakázaná. Povoľovacie pravidlo udeľuje subjektu v danej doméne určené práva voči objektu určeného typu a triedy. Doména subjektu sa nazýva aj *zdrojový typ* (*source type*) pravidla, typ objektu sa nazýva *cieľový typ* (*target type*) pravidla.

Povoľovacie pravidlá majú nasledujúci tvar:

```
allow zdroj cieľ : trieda práva ;
```

Toto pravidlo udeľuje subjektu v doméne *zdroj* voči subjektu typu *cieľ* a triedy *trieda* práva *práva*. Zdrojový aj cieľový typ môžu byť špecifikované viacerými spôsobmi:

- identifikátorom typu – znamená jeden konkrétny typ,
- identifikátorom atribútu typu – znamená akýkoľvek typ, ktorý má priradený uvedený atribút,
- množinou mien typov a atribútov – znamená to akýkoľvek typ, ktorý je uvedený v tejto množine, resp. typ, ktorý má priradený nejaký atribút uvedený v tejto množine.

V prípade, že je typ určený množinou, táto množina sa zapisuje nasledujúcim spôsobom:

```
{typ1_t typ2_t attr1}
```

Pri zápise množiny typov je tiež možné nejaký typ, ktorý je do nej zahrnutý prostredníctvom atribútu, explicitne vylúčiť tým, že sa pred identifikátor typu uvedie operátor –. Ak napríklad typy

typ1_t a *typ2_t* majú priradený atribút *attr1*, tak množina

```
{attr1 -typ1_t}
```

nebude obsahovať *typ1_t*.

Ak je cieľový typ rovnaký ako zdrojový, môže sa cieľový typ nahradiť kľúčovým slovom *self*. To sa využíva, napríklad, pri povoľovaní operácií procesu so sebou samým.

Trieda objektu je špecifikovaná svojím identifikátorom, prípadne množinou identifikátorov:

```
file
{file dir}
```

Práva môžu byť špecifikované identifikátorom, množinou identifikátorov alebo znakom *, ktorý znamená „všetky práva príslušnej triedy“. Použitím operátora ~ pred špecifikáciou práv je možné množinu práv negovať, teda špecifikovať práva ako „všetky práva okrem uvedených“. Príklady špecifikácie práv teda môžu vyzerať nasledovne:

```
read
{read write search}
*
~read
```

SELinux štandardne generuje auditné záznamy o každej zamietnutej operácii. Zvyčajne je to užitočná vlastnosť, pretože umožňuje zistiť, že sa nejaký proces pokúšal vykonať nepovolené operácie. Ak však vieme, že niektoré sa procesy pri svojej normálnej činnosti pokúšajú o niektoré operácie, ktoré nemajú povolené, môže byť vhodné auditné záznamy o týchto zamietnutých operáciách vylúčiť. To umožňuje pravidlo *dontaudit*, ktoré síce operáciu nepovolí, ale potlačí generovanie auditného záznamu o jej zamietnutí:

```
dontaudit zdroj cieľ : trieda práva ;
```

Význam všetkých parametrov je rovnaký ako pri pravidle *allow*.

O povolených operáciách SELinux štandardne auditné záznamy negeneruje. Môžeme však použiť pravidlo *auditallow*, ktoré spôsobí vytvorenie auditného záznamu aj pri povolenej operácii:

```
auditallow zdroj cieľ : trieda práva ;
```

Pravidlo *auditallow* ovplyvňuje len generovanie auditného záznamu, nespôsobí povolenie operácie. Preto má význam len pri súčasnom použití príslušného pravidla *allow*, ktoré príslušné operácie

povolí.

Okrem uvedených troch typov pravidiel je možné v politike použiť ešte pravidlo *neverallow*. Toto pravidlo slúži na znemožnenie toho, aby politika povolila nejakú operáciu. Ak sa pri kompilácii politiky do výsledného binárneho tvaru zistí, že obsahuje povoľovacie pravidlo, ktoré je v rozpore s pravidlom *neverallow*, kompilácia skončí s chybou. Týmto spôsobom môžeme zabezpečiť, že nejaký prístup nebude povolený, ani keby sme neskôr omylom pridali pravidlo, ktoré by ho povolilo. To má význam najmä pri veľkých, zložitých politikách, kedy nemusí byť jednoduché všetky súvislosti a dôsledky skontrolovať ručne.

5.3.3. Triedy objektov a práva

SELinux podporuje pomerne veľké množstvo tried objektov. V súčasnosti je v Linux-ovom jadre definovaných cca 50 tried, ako napr. *file, dir, filesystem, process, packet, security, capability, ….* Každá trieda ma definované svoje špecifické práva, ktoré sú jadrom systému ako správcom objektov kontrolované pri rôznych operáciách. Zoznam tried objektov a zoznam práv pre jednotlivé triedy je možné jednoducho zobraziť využitím pseudosúborového systému *selinuxfs*. Tento súborový systém je štandardne pripojený na adresári */sys/fs/selinux*, a obsahuje adresár *class*, v ktorom sa nachádzajú podadresáre nazvané identifikátormi jednotlivých podporovaných tried. V nich sa nachádza podadresár *perms*, ktorý obsahuje súbory nazvané identifikátormi jednotlivých práv príslušnej triedy. Na nasledujúcich obrázkoch môžeme vidieť zoznam práv pre niektoré vybrané triedy.

```
root@debian:/sys/fs/selinux/class# ls process/perms
dyntransition  getcap      ptrace       setkeycreate   sigchld    transition
execheap       getpgid     rlimitinh    setpgid        siginh
execmem        getrlimit   setcap       setrlimit      sigkill
execstack      getsched    setcurrent   setsched       signal
fork           getsession  setexec      setsockcreate  signull
getattr        noatsecure  setfscreate  share        sigstop
```

Práva pre procesy zahŕňajú širokú množinu operácií, ktoré môže proces aplikovať sám na seba alebo na iné procesy, ako napr. zisťovanie atribútov (*get**), nastavovanie rôznych atribútov (*set**), posielanie signálov (*sig**), vytváranie nových procesov (*fork*), prechod do inej domény pri spustení programu (*transition*). Pri posielaní signálov si môžeme všimnúť, že niektoré signály sú kontrolované vlastným právom, ostatné spoločne právom *signal*. Dôvodom je fakt, že niektoré signály (ako napr. SIGCHLD informujúci rodičovský proces o ukončení „dieťaťa“) si mnohé procesy musia vedieť poslať za bežných okolností, a to aj vtedy, keď posielanie iných signálov medzi nimi nie je žiadúce.

Na nasledujúcich obrázkoch vidíme práva aplikovateľné na súbory (*file*) a adresáre (*dir*).

```
root@debian:/sys/fs/selinux/class# ls file/perms
append        execmod             ioctl  mounton      relabelfrom  swapon
audit_access  execute             link   open         relabelto    unlink
create        execute_no_trans    lock   quotaon      rename       write
entrypoint    getattr             map    read         setattr

root@debian:/sys/fs/selinux/class# ls dir/perms
add_name      execmod  link       open         relabelto    rmdir    unlink
append        execute  lock       quotaon      remove_name  search   write
audit_access  getattr  map        read         rename       setattr
create        ioctl    mounton    relabelfrom  reparent     swapon
```

Ako môžeme vidieť, práva na súbory sú členené jemnejšie ako tradičné prístupové práva. Pri súboroch a adresároch vieme napr. kontrolovať aj právo na vytvorenie (*create*), vytvorenie nového odkazu (*link*), vymazanie (*unlink*), premenovanie (*rename*), správu zámkov (*lock*), a pod. Pri adresároch vieme, okrem zápisu ako takého (*write*) špeciálne kontrolovať napr. pridávanie (*add_name*) a odoberanie (*remove_name*) položiek. Práva *relabelfrom* a *relabelto* umožňujú kontrolovať, či má proces právo na zmenu bezpečnostného kontextu objektu z, resp. na uvedený kontext. Práva *execute*, *execute_no_trans* a *entrypoint* sa používajú na kontrolu spúšťania programov a bližšie sa s nimi oboznámime neskôr.

Aj súborové systémy sú predmetom kontroly SELinux-u:

```
root@debian:/sys/fs/selinux/class# ls filesystem/perms
associate     mount        quotamod     relabelto    transition
getattr       quotaget     relabelfrom  remount      unmount
```

Z práv na súborové systémy stojí za zmienku právo *associate*, ktoré určuje, či objekt zdrojového typu môže byť vytvorený v súborovom systéme cieľového typu. Tento prípad je zvláštny tým, že zdrojovým typom nie je doména procesu, ale typ vytváraného objektu.

Špeciálnou triedou objektu je trieda *security*, ktorá slúži na kontrolu operácií so subsystémom SELinux ako celkom. Typickým príkladom je nahratie politiky do jadra systému (*load_policy*):

```
root@debian:/sys/fs/selinux/class# ls security/perms
check_context  compute_member       load_policy setcheckreqprot  validate_trans
compute_av     compute_relabel      read_policy  setenforce
compute_create compute_user         setbool      setsecparam
```

Ako sme už spomenuli, SELinux kontroluje aj používanie oprávnení subsystému *capabilities*:

```
root@debian:/sys/fs/selinux/class# ls capability/perms
audit_control   ipc_lock           net_bind_service    sys_admin    sys_rawio
audit_write     ipc_owner          net_broadcast       sys_boot     sys_resource
chown           kill               net_raw             sys_chroot   sys_time
dac_override    lease              setfcap             sys_module   sys_tty_config
dac_read_search linux_immutable    setgid              sys_nice
fowner          mknod              setpcap             sys_pacct
fsetid          net_admin          setuid              sys_ptrace
```

Triedy a práva spomenuté v tejto časti sú len ilustratívnym príkladom. Bližšie informácie o význame jednotlivých práv je možné nájsť napr. na http://selinuxproject.org/.

5.3.4. Odvodzovanie typov

Pri vytváraní nového objektu je potrebné inicializovať jeho bezpečnostný kontext. Subjekt môže explicitne špecifikovať, aký bezpečnostný kontext má nový objekt mať. To však robia len programy, ktoré sú vytvorené špeciálne pre použitie v systéme s aktivovaným subsystémom SELinux, a aj to len v prípde, keď je takáto explicitná špecifikácia bezpečnostného kontextu pre nový objekt potrebná. V bežných prípadoch teda musí nový bezpečnostný kontext určiť bezpečnostný server SELinux-u automaticky. Postup závisí v prvom rade od triedy nového objektu.

V prípade objektov súborového systému sa používateľ nového objektu zdedí od vytvárajúceho procesu, rola na nastaví na *object_r* a typ sa zdedí z adresára, v ktorom je objekt vytvorený. Rozsah MLS značiek sa nastaví na dolnú MLS značku vytvárajúceho procesu. Ako uvidíme ďalej, toto štandardné správanie môže byť zmenené použitím pravidla *type_transition* v politike.

V prípade objektov prirodzene naviazaných na proces (napr. deskriptory súborov (*file descriptors*), socket-y, zdieľaná pamäť, …) sa bezpečnostný kontext zdedí z vytvárajúceho procesu.

V prípade vytvorenia nového procesu (*fork*) sa bezpečnostný kontext zdedí z vytvárajúceho procesu.

Dôležitým prípadom je spustenie nového programu (*execve*). Nový program môže byť spustený buď v pôvodnej doméne procesu (v tomto prípade k žiadnej zmene nedochádza), alebo môže jeho spustením dôjsť k zmene domény. Ak nie je v politike určené pravidlom *type_transition* inak a proces nepožiadal explicitne o zmenu domény, k zmene domény nedôjde. V tom prípade pôvodná doména potrebuje voči typu súboru so spúšťaným programom práva *execute* a *execute_no_trans*. Právo *execute* oprávňuje doménu spustiť program z tohto súboru a právo *execute_no_trans* povoľuje doméne spustiť taký program bez zmeny domény. To umožňuje zabezpečiť, že proces bežiaci v určitej doméne nemôže v tejto doméne spúšťať iné programy, a teda nemôže delegovať svoje oprávnenia na iné programy. Zároveň mu to ale nemusí znemožniť spúšťanie programov so zmenou domény.

Na to, aby proces spustil program s prechodom do inej domény, musí platiť:

- pôvodná doména má právo *execute* voči typu súboru s programom,
- nová doména má právo *entrypoint* voči typu súboru s programom a
- pôvodná doména má právo *transition* voči novej doméne.

Keď sú tieto podmienky splnené, proces môže explicitne určiť novú doménu, v ktorej má byť program spustený, alebo to môže byť určené implicitne politkou pomocou pravidla *type_transition*.

Ako vidíme, pravidlo *type_transition* sa používa na definovanie nového typu

- pre nový objekt súborového systému,
- pre proces pri spustení nového programu.

Toto pravidlo má nasledujúcu podobu:

```
type_transition zdroj ciel : trieda novy ;
```

Účel použitia a význam pravidla je určený triedou objektu:

- *process*
  - ak proces v doméne *zdroj* spustí program zo súboru typu *ciel*, nová doména bude *novy*,
- *file, dir, lnk_file, ...* (triedy objektov súborového systému)
  - ak proces v doméne *zdroj* vytvorí nový objekt príslušnej triedy v adresári typu *ciel*, nový objekt bude mať typ *novy*.

Politika SELinx-u môže obsahovať aj podobné pravidlo:

```
type_change zdroj ciel : trieda novy ;
```

Toto pravidlo slúži podporným programom (napr. pri prihlasovaní používateľa) na určenie nového typu niektorých objektov, ktorých bezpečnostný kontext je potrebné prispôsobiť používateľovi. Typickým príkladom je znakové zariadenie reprezentujúce terminál, na ktorom sa používateľ prihlásil.

5.3.5. Roly

Každá rola v SELinux-e má svoj identifikátor. Bežná konvencia pri pomenovávaní rol v SELinux-e používa príponu *_r* na označenie, že identifikátor označuje rolu. Rola sa definuje deklaráciou nasledujúceho tvaru:

```
role id_r ;
```

Následne potom môžeme role priradiť povolené domény. Na to slúži deklarácia v nasledujúcich tvarov:

```
role id_r types dom_t ;
role id_r types {typ1_t attr2 -typ2_t} ;
```

Prvá deklarácia priradí role *id_r* povolenú doménu *dom_t*. Druhá jej priradí niekoľko povolených domén – množina typov môže byť určená podobne, ako pri povoľovacích pravidlách. V tomto

konkrétnom príklade teda role *id_r* priradí ako povolené domény *typ1_t*, všetky typy s atribútom *attr2* okrem typu *typ2_t*. Pokiaľ politika obsahuje viacero deklarácií na priradenie domén rovnakej role, rola bude mať priradené zjednotenie domén zo všetkých takých deklarácií.

Aby bolo možné realizovať zmenu roly, musí politika takúto zmenu povoľovať. Na povolenie prechodu medzi rolami slúži povoľovacie pravidlo tvaru:

```
allow stara_r nova_r ;
```

ktoré povoľuje prechod z roly *stara_r* do roly *nova_r*.

K zmene roly môže dôjsť buď na základe explicitnej požiadavky, alebo aj automaticky pri spustení programu. Na automatickú zmenu roly slúži pravidlo:

```
role_transition stara_r typ_t nova_r ;
```

Toto zabezpečí, že ak proces konajúci v role *stara_r* spustí program typu *typ_t*, tak sa má rola zmeniť na *nova_r*. Samozrejme, taká zmena musí byť aj politikou povolená, inak spustenie takého programu nebude možné.

5.3.6. Používatelia

Bezpečnostná politika SELinux-u definuje aj identifikátory používateľov. Bežná konvencia používa príponu *_u* na označenie, že identifikátor označuje používateľa. Používateľ musí mať pridelenú aspoň jednu povolenú rolu. Deklarácia používateľa má jeden z nasledujúcich tvarov:

```
user id_u roles id_r ;

user id_u roles {id1_r id2_r} ;
```

V prvom prípade definujeme používateľa *id_u* s jednou povolenou rolou, v druhom prípade s viacerými povolenými rolami.

Ak politika podporuje MLS, tak je potrebné navyše špecifikovať rozsah MLS značiek pre používateľa a počiatočnú (*default*) úroveň. Deklarácia má v takom prípade ešte ďalšie parametre ako v nasledujúcom príklade:

```
user id_u roles id_r level s0 range s0-s2:c0.c5;
```

Používateľov je možné do systému dopĺňať aj nad rámec tých, ktorí sú definovaní priamo v politike pomocou nástrojov na správu politiky, ktorým sa budeme venovať neskôr.

5.3.7. Obmedzenia

Okrem povolenia operácií (pomocou pravidla *allow*) kontroluje SELinux operácie ešte aj pomocou

tzv. obmedzení. Obmedzenia sa vzťahujú vždy na triedu objektu a požadované práva a určujú podmienky, ktoré musia byť splnené, aby uvedené práva na objekt danej triedy mohli byť žiadajúcemu subjektu udelené. Podmienky je možné špecifikovať formou logického výrazu, v ktorom je možné porovnávať používateľa, rolu a/alebo typ subjektu a objektu navzájom alebo so zoznamom. Obmedzenia sa používajú napríklad na dosiahnutie toho, že používateľ nemôže uplatňovať práva na objekty iného používateľa, teda že používateľ subjektu a objektu musí byť rovnaký. Taktiež je možné obmedzenia využiť na to, aby používatelia na zmenu roly museli použiť konkrétny program, ktorý napr. zabezpečí ich autentifikáciu.

Obmedzenia sa definujú deklaráciou v nasledujúcom tvare:

```
constrain trieda práva výraz ;
```

kde *trieda* definuje jednu alebo viac tried, *práva* je jedno alebo viac práv, na ktoré sa má obmedzenie aplikovať (ak je tried alebo práv viac, uvedú sa v {}) a *výraz* je logický výraz, ktorý musí byť na udelenie práv splnený. Tento výraz môže obsahovať zátvorky, logické spojky *and*, *or*, a *not*. Môže obsahovať porovnania pomocou operátorov == (rovnosť) a != (nerovnosť) medzi:

- *u1* a *u2*,
- *r1* a *r2*,
- *t1* a *t2*,
- *u1* a zoznamom používateľov,
- *u2* a zoznamom používateľov,
- *r1* a zoznamom rol,
- *r2* a zoznamom rol,
- *t1* a zoznamom typov,
- *t2* a zoznamom typov.

Premenné *u1* a *u2* obsahujú používateľa, *r1* a *r2* rolu a *t1* a *t2* typ (doménu) zdrojového (1) a cieľového (2) bezpečnostného kontextu. Zoznam používateľov, rol a typov môže obsahovať jeden identifikátor alebo zoznam identifikátorov v {}. V prípade typov je možné použiť aj atribúty typov. Pri použití porovnávacích operátorov oproti zoznamu má operátor == význam operácie „je prvkom množiny“ a operátor != význam „nie je prvkom množiny“.

Obmedzenie môže vyzerať napríklad nasledovne:

```
constrain {file dir lnk_file} { create relabelto relabelfrom }
(
        u1 == u2
        or t1 == can_change_object_identity
);
```

Toto obmedzenie zabzepečí, že pri práva *create, relabelfrom* a *relabelto* môžu byť udelené len za predpokladu, že používateľ zdrojového procesu (*u1*) je rovnaký ako používateľ objektu (*u2*) triedy *file*, *dir*, alebo *lnk_file* alebo doména procesu (*t1*) musí mať atribút *can_change_object_identity*. Inými slovami, ak proces nemá špeciálne oprávnenie (vyjadrené doménou s určitým atribútom), nemôže vytvoriť objekt s priradeným iným používateľom, nemôže zmeniť používateľa objektu, ak nie je jeho vlastný, a nemôže zmeniť používateľa objektu na iného ako seba.

5.3.8. MLS značky a obmedzenia

Ako sme spomenuli už skôr, komponent MLS je v SELinux-e pomerne flexibilne konfigurovateľný. Politika musí najprv definovať identifikátory jednotlivých úrovní, ich usporiadanie a identifikátory jednotlivých kategórií. Následne musí definovať prípustné kombinácie úrovne a kategórií, teda nie je nutne potrebné, aby na každej úrovni boli prípustné všetky kategórie.

Úrovne sa definujú deklaráciou *sensitivity*:

```
sensitivity s0;
sensitivity s1;
sensitivity s2;
```

Usporiadanie úrovní sa definuje deklaráciou *dominance*, zoznam úrovní je od najnižšej po najvyššiu:

```
dominance {s0 s1 s2}
```

Kategórie sa definujú deklaráciou *category*:

```
category c0;
category c1;
category c2;
```

Povolené kombinácie úrovne a množiny kategórií pre bezpečnostné značky sa definujú deklaráciami *level*. Pre každú úroveň musí politika obsahovať práve jednu takúto deklaráciu:

```
level s0:c0;
level s1:c0,c1;
level s2:c0.c2;
```

V uvedenom príklade môže byť množina kategórií bezpečnostnej značky s úroveňou *s0* len prázdna

alebo môže obsahovať *c0*, množina bezpečnostnej značky s úrovňou *s1* môže obsahovať aj kategóriu *c1* a množina bezpečnostnej značky s úrovňou *s2* môže obsahovať akúkoľvek podmnožinu kategórií.

Ďalšou dôležitou súčasťou politiky pre použitie MLS je definícia pravidiel, ktoré implementujú požadovaný MLS bezpečnostný model. Na to slúžia obmedzenia definované pomocou deklarácie *mlsconstrain*, ktoré sa rovnako ako obmedzenia *constrain* viažu na triedu a práva a určujú ďalšie podmienky, ktoré musia byť splnené, aby boli práva udelené. Táto deklarácia má tvar:

```
mlsconstrain trieda práva výraz ;
```

a výraz môže obsahovať všetko, čo výraz deklarácie *constrain*, a navyše môže obsahovať porovnania medzi bezpečnostnými značkami:

- *l1* a *l2*,
- *l1* a *h2*,
- *h1* a *l2*,
- *h1* a *h2*,
- *l1* a *h1*,
- *l2* a *h2*,

kde *l1* a *l2* je dolná a *h1* a *h2* horná bezpečnostná značka zdrojového (1) a cieľového (2) bezpečnostného kontextu. Pri porovnávaní bezpečnostných značiek je možné použiť (okrem == a ! =) aj operátory:

- *l1 dom l2*
  - *l1* dominuje (je nad) *l2*, teda $l1 \geq l2$,
- *l1 domby l2*
  - *l1* je dominovaná (je pod) *l2*, teda $l1 \leq l2$,
- *l1 incomp l2*
  - *l1* a *l2* sú neporovnateľné.

Pomocou takýchto obmedzení je možné popísať napríklad pravidlá Bell – La Padula, Biba, alebo iných podobných modelov:

```
mlsconstrain file { read ioctl lock execute execute_no_trans }
```

```
  (( h1 dom h2 ) or ( t1 == mcsreadall ) or ( t2 == domain ));

mlsconstrain file { write setattr append link rename }
  ((( h1 dom h2 ) and ( l1 domby l2 )) or ( t1 == mcswriteall ) or (t2 ==
mcstrustedobject) or ( t2 == domain ));
```

Uvedený príklad zabezpečí, že pre čítanie zo súboru musí byť horná bezpečnostná značka procesu nad (hornou) bezpečnostnou značkou objektu alebo musia byť splnené podmienky pre výnimky, a že pre zápis musí byť dolná bezpečnostná značka procesu pod (dolnou) bezpečnostnou značkou objektu a horná bezpečnostná značka procesu nad (hornou) bezpečnostnou značkou objektu alebo musia byť splnené podmienky pre výnimky. Toto konkrétne obmedzenie definuje slabšie pravidlá ako spomínané modely, ale aj tak limituje proces v tom, ako „hlboko“ môže vyniesť citlivú informáciu.

Politika tiež umožňuje definovať pravidlo, ktoré určuje nový rozsah MLS značiek pri inicializácii bezpečnostného kontextu podobne, ako pri určovaní nového typu pomocou *type_transition* pravidla. Slúži na to pravidlo:

```
range_transition zdroj cieľ : trieda nový_rozsah ;
```

Pomocou neho je možné napr. nastaviť nový rozsah MLS značiek pri spustení programu typu *cieľ* z domény *zdroj*.

5.3.9. Špecifikácia bezpečnostných kontextov niektorých objektov

Niektoré objekty, ako súborové systémy, sieťové rozhrania, sieťové adresy, či sieťové porty majú priradený bezpečnostný kontext pomocou špeciálnych pravidiel v politike. Asi najpoužívanejším z nich je pravidlo pre pridelenie bezpečnostného kontextu sieťovým portom, pretože ten sa často využíva na obmedzenie toho, aké porty môže proces v určitej doméne používať. Priradenie bezpečnostného kontextu portom sa realizuje deklaráciou v nasledujúcom tvare:

```
portcon protokol port kontext
```

Parameter protokol obsahuje *tcp* alebo *udp*, *port* je číslo portu, prípadne rozsah v tvare *od-do* a *kontext* je bezpečnostný kontext. Napríklad:

```
portcon tcp 20 system_u:object_r:ftp_data_port_t
portcon tcp 21 system_u:object_r:ftp_port_t
portcon tcp 600-1023 system_u:object_r:hi_reserved_port_t
portcon udp 600-1023 system_u:object_r:hi_reserved_port_t
portcon tcp 1-599 system_u:object_r:reserved_port_t
portcon udp 1-599 system_u:object_r:reserved_port_t
```

Použitie portov sa povoľuje štandardným povoľovacím pravidlo *allow*, pričom ako zdrojový typ sa

použije doména procesu, ako cieľový typ sa použije typ portu a ako trieda sa použije *tcp_socket* alebo *udp_socket*. Pre oba typy socket-ov je možné prideľovať právo *name_bind*, ktoré kontroluje použitie portu na lokálnej strane, pre TCP aj právo *name_connect*, ktoré kontroluje použitie vzdialeného portu pri volaní *connect*.

5.4. Referenčná politika

Ako je možné tušiť na základe predchádzajúcich častí, vytvorenie kompletnej SELinux-ovej politiky pre komplexný systém nie je jednoduchá vec. Našťastie, bola už vytvorená tzv. *referenčná politika*, ktorá pokrýva veľa bežných služieb a aplikácií na Linux-ových systémoch. Vývoj tejto politiky začal v podobe vhodnej pre servery, ktorá maximálne obmedzovala všetky služby a používateľov. Táto politika bola známa aj ako tzv. *striktná (strict)*. Neskôr prišla snaha o úpravu politiky tak, aby bola použiteľná aj na pracovné stanice. Kľúčovým problémom sa ukázala príliš veľká variabilita požiaviek a očakávaní používateľov pracovných staníc v porovnaní so systémom pre servery. Viedlo to k vytvoreniu tzv. *cielenej (targeted)* politiky, ktorá neobmedzovala používateľa a jeho procesy, ale obmedzovala systémové služby. Neskôr došlo k spojeniu striktnej a cielenej politiky do jedného celku, ktorý je možné nakonfigurovať oboma spôsobmi, resp. aj pre rôznych používateľov rôzne.

Z hľadiska štruktúry SELinux-ovej politiky a procesu jej transformácie do výsledného binárneho tvaru došlo tiež k významnému vývoju. Na začiatku bolo potrebné vytvoriť veľký textový súbor popisujúci celú politiku, ktorý sa následne kompiloval do binárnej podoby. Dnes je politika *modulárna*, teda skladá sa zo samostatných modulov, ktoré sa samostatne kompilujú z textového popisu do binárnej podoby – tzv. balíčkov (*policy packages*). Vybrané binárne balíčky je následne možné skombinovať do výslednej binárnej politiky, ktorá sa nahráva do jadra systému. To umožňuje vysokú variabilitu politiky bez potreby jej kompletnej novej kompilácie pri každej zmene.

Referenčná politika sa aktuálne skladá z vyše 300 modulov, jej výsledná textová podoba má cca 4 milióny riadkov. Jeden modul typicky pokrýva jednu službu alebo aplikáciu. Keďže moduly je možné jednoducho do aktívnej politiky prídávať a odoberať, umožňuje to nakonfigurovať politiku flexibilne podľa toho, aké služby či aplikácie máme v systéme nainštalované.

Modul politiky definuje svoje vlastné typy, atribúty typov, roly a pravidlá (povoľovacie, auditovacie, pre odvodzovanie typov). Nemôže definovať obmedzenia – tie musia byť v základe (jadre) politiky. Okrem toho modul definuje rozhrania (*interface*), prostredníctvom ktorých dáva možnosť iným modulom využiť svoje služby. Tieto rozhrania v podstate predstavujú

parametrizované šablóny deklarácií v jazyku politiky, ktoré môžu iné moduly využiť (po substitúcii parametrov). Z dôvodu udržateľnosti poriadku v tak rozsiahlom projekte, akým referenčná politika je, je totiž nežiadúce, aby mohli jednotlivé moduly ľubovoľne používať typy a atribúty definované v inom module. Preto sú všetky typy, atribúty aj roly definované modulom považované za „súkromné“, a jediný korektný spôsob, ako ich môžu iné moduly využiť, je prostredníctvom rozhrania, ktoré im modul poskytuje. Modul tiež definuje bezpečnostné kontexty, ktoré majú byť priradené jednotlivým adresárom a súborom v adresárovom strome.

V nasledujúcich častiach sa pozrieme na štandardné roly a používateľov definovaných v referenčnej politike.

5.4.1. Roly v referenčnej politike

Referenčná politika definuje niekoľko štandardných rol:

- *user_r*
  - je rola pre bežného používateľa, ktorý nemá právo na administráciu systému,
- *staff_r*
  - je neprivilegovaná rola pre administrátora systému, teda rola, v ktorej administrátor nemá administrátorské oprávnenia, ale len oprávnenia podobné role *user_r*; môže však použiť príkaz *newrole* na zmenu roly na *sysadm_r*,
- *sysadm_r*
  - je privilegovaná rola pre administrátora, teda rola, v ktorej má administrátor administrátorské oprávnenia,
- *system_r*
  - je vyhradená rola pre systémové procesy,
- *unconfined_r*
  - je rola pre neobmedzeného používateľa – táto slúži na implementáciu cielenej politiky; používa sa pre používateľa, ktorý nemá takmer žiadne obmedzenia.

5.4.2. Používatelia v referenčnej politike

Referenčná politiky definuje niekoľko štandardných používateľov:

- *system_u*
  - je špeciálny používateľ určený pre systémové procesy a objekty, nebýva priradený žiadnemu skutočnému používateľovi,
  - má povolenú rolu: *system_r*,
- *user_u*
  - je bežný používateľ bez administrátorských oprávnení, typicky sa používa ako SELinux-ová identita pre všetkých bežných používateľov,
  - má povolenú rolu *user_r*,
- *staff_u*
  - je administrátor, ktorý má po prihlásení bežné používateľské práva, ale môže si zmeniť rolu na *sysadm_r*, čím administrátorské práva získa,
  - má povolené roly *staff_r* a *sysadm_r*,
- *sysadm_u*
  - je administrátor, ktorý má administrátorské práva automaticky hneď pri prihlásení; nie je vhodné túto identitu používať na bežnú činnosť, ktorá s administráciou systému nesúvisí,
  - má povolenú rolu *sysadm_r*,
- *unconfined_u*
  - je (takmer) neobmedzený používateľ z pohľadu SELinux-u; táto identita sa používa pre používateľov v cielenom režime politiky,
  - má povolené roly *unconfined_r* a *system_r*,
- *root*
  - je SELinux-om obmedzený root – identita určená pre Linux-ového používateľa root pri použití striktného režimu politiky,
  - má povolené roly *staff_r*, *sysadm_r*, *system_r*.

Za bežných okolností je referenčná politika použiteľná bez pridávania ďalších SELinux-ových používateľov. Vyššie uvedené SELinux-ové identity sa priraďujú rôznym Linux-ovým používateľom podľa toho, či majú alebo nemajú byť schopní spravovať systém. Ak má niektorý Linux-ový používateľ priradenú SELinux-ovú identitu *unconfined_u*, SELinux ho takmer nebude obmedzovať. Ak majú túto identitu priradenú všetci reálni používatelia systému, získavame stav zodpovedajúci cielenej SELinux-ovej politike. Ak nebude mať túto identitu priradenú žiadny používateľ, získame stav zodpovedajúci striktnej SELinux-ovej politike.

5.5. Správa a konfigurácia SELinux-u

V tejto časti sa pozrieme na nástroje na správu a používanie SELinux-u. Taktiež sa pozrieme na niektoré konfiguračné súbory SELinux-u.

5.5.1. Správa používateľov

Jedným zo základných aspektov správy používateľov SELinux-u je správa mapovania medzi Linux-ovými používateľmi a priradenými SELinux-ovými identitami. Na tieto účely (a ďalšie) účely slúži príkaz *semanage*:

```
semanage login oper [opts] login_name
```

Prvý parameter príkazu (*login*) určuje, či chceme týmto univerzálnym príkazom spravovať. V našom prípade je to mapovanie prihlasovacích mien na SELinux-ové identity. Parameter *oper* určuje operáciu, ktorú chceme vykonať:

- *-a* – pridanie záznamu,
- *-m* – zmena existujúceho záznamu,
- *-d* – vymazanie existujúceho záznamu,
- *-l* – zobrazenie všetkých záznamov,
- *-D* – vymazanie všetkých lokálnych zmien.

Tieto operácie neplatia len pre správu mapovania prihlasovacích mien na SELinux-ové identity, ale všeobecne pre príkaz *semanage*.

Posledný parameter príkazu (*login_name*) je Linux-ové prihlasovacie meno používateľa, s ktorého záznamom chceme manipulovať. Pri pridávaní a zmene existujúceho záznamu sú potrebné ďalšie parametre (*opts*), pomocou ktorých sa špecifikujú ďalšie atribúty záznamu. V prípade správy mapovania identít sú to:

- *-s selinux_name*
    - SELinux-ový používateľ, ktorý má byť Linux-ovému používateľovi priradený,
- *-r mls_range*
    - rozsah MLS značiek, ktorý má byť používateľovi nastavený pri prihlásení.

Použitie príkazu *semanage* na správu mapovania identít si môžeme demonštrovať na nasledujúcom príklade:

```
root@debian:~# semanage login -a -s staff_u  admin
root@debian:~# semanage login -a -s root -r s0-s0:c0.c10 root2

root@debian:~# semanage login -l
Login Name                SELinux User              MLS/MCS Range
__default__               unconfined_u              s0-s0:c0.c1023
admin                     staff_u                   s0
luser                     user_u                    s0
root                      unconfined_u              s0-s0:c0.c1023
root2                     root                      s0-s0:c0.c10
system_u                  system_u                  s0-s0:c0.c1023
```

Prvým príkazom sme definovali, že používateľ *admin* má mať priradenú SELinux-ovú identitu *staff_u*, teda bude môcť získať administrátorské oprávnenia zmenou roly. Následne sme pridali používateľovi *root2* SELinux-ovú identitu *root* s uvedením rozsahu MLS značiek od *s0* po *s0:c0.c10*. Posledným príkazom sme zobrazili celé mapovanie identít. Vo výpise si môžeme všimnúť záznam s prihlasovacím menom *__default__*. Tento sa použije pre všetkých používateľov, pre ktorých nie je definovaný explicitný záznam. Ako vidíme, v našom prípade budú mať všetci explicitne neuvedení používatelia priradenú identitu *unconfined_u* s plným rozsahom MLS značiek. Je to typický príklad systému s cielenom (*targeted*) politikou – bežní používatelia nie sú obmedzovaní. Keby sme tento záznam zmenili napr. na *user_u*, všetci neuvedení používatelia by boli štandardným SELinux-ovým bežným používateľom.

Ako sme spomenuli v časti venovanej jazyku SELinux-ovej politiky, SELinux-oví používatelia môžu byť definovaní priamo v politike, ale môžu byť doplnení aj dodatočne. Na to slúži opäť príkaz *semanage*, tento raz s prvým parametrom *user*:

```
semanage user oper [opts] selinux_name
```

Posledným parametrom je teraz SELinux-ový identifikátor používateľa, operácie *oper* sú rovnaké ako pre *semanage login*. Voľby *opts*, ktorými sa nastavujú ďalšie atribúty záznamu sú nasledovné:

- *-L level*
    - štandardná MLS úroveň pre používateľa,

- *-P prefix*
    - prefix používaný pre značkovanie domovských adresárov; táto hodnota sa substituuje do šablóny na generovanie bezpečnostných kontextov pre domovské adresáre, pri použití referenčnej politiky je štandardne rovnaká ako rola používateľa bez prípony *_r*,
- *-r mls_range*
    - rozsah MLS značiek pre SELinux-ového používateľa,
- -R role
    - povolená rola pre používateľa (je možné uviesť aj viac rol oddelených medzerami a spoločne uzavretých v úvodzovkách, prípadne je tento prepínač možné použiť viackrát).

Pomocou tohto príkazu môžeme pridávať a odoberať SELinux-ových používateľov bez toho, aby sme museli upraviť zdrojový text politiky a znovu ju kompilovať. Taktiež môžeme meniť povolené roly a MLS rozsahy používateľov.

Použitie príkazu si môžeme opäť demonštrovať na príklade:

```
root@debian:~# semanage user -a -P user -R user_r test_u

root@debian:~# semanage user -l

                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range           SELinux Roles

root            sysadm     s0         s0-s0:c0.c1023      staff_r sysadm_r system_r
staff_u         staff      s0         s0-s0:c0.c1023      staff_r sysadm_r
sysadm_u        sysadm     s0         s0-s0:c0.c1023      sysadm_r
system_u        user       s0         s0-s0:c0.c1023      system_r
test_u          user       s0         s0                  user_r
unconfined_u    unconfined s0         s0-s0:c0.c1023      system_r unconfined_r
user_u          user       s0         s0                  user_r
```

Prvým príkazom sme pridali nového SELinux-ového používateľa *test_u*, ktorý bude mať priradenú rolu *user_r*. Druhým príkazom sme zobrazili zoznam všetkých definovaných SELinux-ových používateľov.

5.5.2. Vybrané konfiguračné súbory SELinux-u

Základným konfiguračným súborom SELinux-u, ktorý obsahuje globálne nastavenia, je súbor

```
/etc/selinux/config
```

Tento súbor obsahuje dva dôležité parametre:

- názov politiky,
- mód – môže mať hodnoty *permissive* alebo *enforcing*.

SELinux môže pracovať v dvoch módoch. V móde *permissive* v skutočnosti nezakazuje žiadne operácie, ale len generuje auditné záznamy o zamietnutí operácií, ktoré nie sú politkou povolené. V *enforcing* móde už SELinux pracuje štandardne, teda nepovolené operácie zakazuje. *Permissive* mód je určený na testovacie účely, najmä pri ladení novej politiky. V reálnej prevádzke, ak chceme získať bezpečnostné výhody, ktoré SELinux prináša, je potrebné použiť *enforcing* mód.

Ďalšia konfigurácia politiky (vrátane binárnej podoby bezpečnostnej politiky a aktívnych modulov) je uložená v adresári

`/etc/selinux/meno_politiky/`

Takáto štruktúra nám umožňuje mať v systéme nainštalovaných viac SELinux-ových politík a prepínať medzi nimi jednoducho zmenou mena aktívnej politiky v */etc/selinux/config*. Ďalšie cesty ku konfiguračným súborom budú uvádzané relatívne k adresáru s konfiguráciou konkrétnej politiky.

Keď sa používateľ prihlasuje do systému, je potrebné určiť bezpečnostný kontext pre jeho prvý proces. Na základe prihlasovacieho mena už vieme zistiť SELinux-ovú identitu používateľa, a rozsah MLS značiek, no chýba nám ešte rola a typ. Prihlasovacie programy, ktoré sú upravené na použitie so SELinux-om na tento účel používajú nasledujúce súbory:

`contexts/users/selinux_user`

`contexts/default_contexts`

Oba súbory majú rovnakú štruktúru, prvý je špecifický pre konkrétneho SELinux-ového používateľa, druhý predstavuje spoločný *default* pre prípad, že sa v špecifickom súbore nenájde potrebná informácia. Súbory obsahujú riadky s niekoľkými medzerami oddelenými trojicami *rola:typ:mls_rozsah*. Prihlasovací program hľadá riadok, kde prvá trojica zodpovedá jeho aktuálnemu bezpečnostnému kontextu. Ak taký riadok nájde, postupne prechádza ďalšie trojice na riadku a vyberie prvú, ktorá po doplnení o používateľa vytvorí platný bezpečnostný kontext. Tento bezpečnostný kontext následne použije pre spúšťaný program (typicky *shell*). Skutočnosť, že sa riadok hľadá na základe bezpečnostného kontextu prihlasovacieho programu, umožňuje používateľovi prideliť rôzny bezpečnostný kontext podľa spôsobu prihlásenia. Môžeme teda rozlíšiť napríklad to, či sa používateľ prihlásil na lokálnej konzole alebo prostredníctvom siete. Tým môžeme napríklad zabezpečiť, že systém nebude možné spravovať na diaľku, ale len pri lokálnom prihlásení. Obsah týchto súborov môže vyzerať napr. nasledovne:

contexts/users/root

```
system_r:crond_t:s0             unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0
system_r:local_login_t:s0       unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0

staff_r:staff_su_t:s0           unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0
sysadm_r:sysadm_su_t:s0         unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0
user_r:user_su_t:s0             unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0

#
# Uncomment if you want to automatically login as sysadm_r
#
#system_r:sshd_t:s0             unconfined_r:unconfined_t:s0 sysadm_r:sysadm_t:s0 staff_r:staff_t:s0 user_r:user_t:s0
```

contexts/default_contexts

```
system_r:crond_t:s0             user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0 system_r:system_cronjob_t:s0
  unconfined_r:unconfined_t:s0
system_r:local_login_t:s0       user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0 unconfined_r:unconfined_t:s0
system_r:remote_login_t:s0      user_r:user_t:s0 staff_r:staff_t:s0 unconfined_r:unconfined_t:s0
system_r:sshd_t:s0              user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0 unconfined_r:unconfined_t:s0
system_r:sulogin_t:s0           sysadm_r:sysadm_t:s0
system_r:xdm_t:s0               user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0 unconfined_r:unconfined_t:s0

staff_r:staff_su_t:s0           user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0
staff_r:staff_sudo_t:s0         sysadm_r:sysadm_t:s0 staff_r:staff_t:s0

sysadm_r:sysadm_su_t:s0         user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0
sysadm_r:sysadm_sudo_t:s0       sysadm_r:sysadm_t:s0

user_r:user_su_t:s0             user_r:user_t:s0 staff_r:staff_t:s0 sysadm_r:sysadm_t:s0
user_r:user_sudo_t:s0           sysadm_r:sysadm_t:s0 user_r:user_t:s0
```

Môžeme si všimnúť napr. rozdiel pri prihlasovaní používateľa *root* na lokálnej konzole (*local_login_t*) a prostredníctvom ssh (*sshd_r*). V prvom prípade sa nájde zvýraznený riadok v prvom súbore (špecifickom pre SELinux-ového používateľa *root*) a vyskúša sa použiť *unconfined_r:unconfined_t:s0*, čo sa napr. pri našom Linux-ovom používatľovi *root2* nepodarí (lebo nemá rolu *unconfined_r* povolenú). Preto sa vyskúša ďalšia možnosť v poradí (*sysadm_r:sysadm_t:s0*) a keďže rolu *sysadm_r* má *root* povolenú, takýto bezpečnostný kontext sa použije.

Ak sa však bude *root2* prihlasovať cez *ssh (sshd_t)*, tak sa v prvom súbore použiteľný riadok nenájde a v druhom (spoločnom) súbore sa nájde zvýraznený riadok. Prvá použiteľná rola na ňom bude *staff_r*, a preto sa táto možnosť použije na určenie bezpečnostného kontextu.

Programy, ktoré umožňujú zmenu roly zas potrebujú získať informáciu o type (doméne), ktorý majú po zmene roly použiť, ak nie je určený explicitne. Na to slúži ďalší konfiguračný súbor:

```
contexts/default_type
```

Ten obsahuje riadky, ktoré k jednotlivým rolám priraďujú *default* typ (doménu):

```
auditadm_r:auditadm_t
secadm_r:secadm_t
sysadm_r:sysadm_t
staff_r:staff_t
unconfined_r:unconfined_t
user_r:user_t
```

Ako vidíme, *default* doménou napr. pre rolu *syadm_r* je *sysadm_t*.

5.5.3. Zmena roly

SELinux-ový používateľ môže mať pridelené viaceré roly. Každý jeho proces však koná vždy v jednej konkrétnej role. Ak chce používateľ spustiť proces (napr. *shell*) s inou rolou, môže na to použiť príkaz *newrole*:

```
newrole [-r rola] [-t domena] [-l mls_range]
```

Tento príkaz spustí nový *shell* v uvedenej role, doméne a s uvedeným rozshom MLS značiek. Ak nie je uvedená doména, príkaz *newrole* určí doménu pomocou informácií v *contexts/default_type*. Príkaz *newrole* si pred spustením *shell*-u vyžiada novú autentifikáciu používateľa, aby predišiel tomu, že ho zneužije nejaký proces bez vedomia používateľa. Prechod zo starej roly do novej roly musí byť, samozrejme, aj povolený politikou a používateľ musí mať novú rolu politiou pridelenú. Ak tieto podmienky nie sú splnené, *newrole* skončí s chybou.

Použitie príkazu *newrole* si môžeme demonštrovať na niekoľkých príkladoch:

```
luser@debian:~$ id
uid=1001(luser) gid=1001(luser) groups=1001(luser)
context=user_u:user_r:user_t:s0

luser@debian:~$ newrole -r sysadm_r
user_u:sysadm_r:sysadm_t:s0 is not a valid context
```

V tomto prípade zmena roly zlyhala, pretože context *user_u:sysadm_r:sysadm_t* je neplatný z dôvodu, že používateľ *user_u* nemá pridelenú povolenú rolu *sysadm_r*.

V ďalšom príklade sme sa prihlásili ako *root2* pomocou *ssh*. SELinux-á identita, ktorú má *root2* priradenú je *root*. Vzhľadom na spôsob prihlásenia mu bola nastavená rola *staff_r* a doména *staff_t*. To znamená, že napriek tomu, že pre tradičný bezpečnostný model má plné root-ovské práva, SELinux bude jeho operácie obmedzovať:

```
root@debian:/home# id
uid=0(root) gid=0(root) groups=0(root)
context=root:staff_r:staff_t:s0:c0-s0:c0.c10

root@debian:/home# ls -la
ls: cannot access luser: Permission denied
ls: cannot access jerry: Permission denied
ls: cannot access admin: Permission denied
total 8
drwxr-xr-x.  5 root root 4096 Oct  8 18:22 .
drwxr-xr-x. 22 root root 4096 Oct 12 19:40 ..
d??????????  ? ?    ?       ?            ? admin
d??????????  ? ?    ?       ?            ? jerry
d??????????  ? ?    ?       ?            ? luser
```

Ako vidíme, tak tento používateľ nebol schopný zistiť ani atribúty domovských adresárov bežných používateľov.

Následne si môže skúsiť zmeniť rolu na *sysadm_r*:

```
root@debian:/home# newrole -r sysadm_r
Password:
root@debian:/home# id
uid=0(root) gid=0(root) groups=0(root)
context=root:sysadm_r:sysadm_t:s0:c0-s0:c0.c10

root@debian:/home# ls -la
total 20
drwxr-xr-x.  5 root  root  4096 Oct  8 18:22 .
drwxr-xr-x. 22 root  root  4096 Oct 12 19:40 ..
drwxr-xr-x.  5 admin admin 4096 Oct  8 22:57 admin
drwxr-xr-x.  2 jerry jerry 4096 Oct  5 18:59 jerry
drwxr-xr-x.  2 luser luser 4096 Oct  8 18:33 luser
```

V tomto prípade bola zmena roly úspešná, keďže kontext *root:sysadm_r:sysadm_t* je platný a prechod medzi rolami *staff_r* a *sysadm_r* je tiež referenčnou politikou povolený.

5.5.4. Zmena kontextu súboru

Bezpečnostné kontexty súborov (a iných objektov súborového systému) v súborových systémoch, ktoré to podporujú, sú uložené v rozšírených atribútoch. Pri inštalácii SELinux-u alebo pri podstatnej zmene bezpečnostnej politiky je potrebné tieto bezpečnostné kontexty inicializovať. Správne bezpečnostné kontexty pre súbory sú definované v politike, a podľa tejto definície ich nastavuje príkaz *restorecon*:

```
restorecon [-R] subor
```

Tento nastaví bezpečnostný kontext objektu *subor* na hodnotu určenú politkou. Prepínač *-R* slúži na rekurzívnu aplikáciu príkazu *restorecon* na celý adresárový podstrom. Na inicializáciu bezpečnostných kontextov celého adresárového stromu je teda možné použiť príkaz:

```
restorecon -R /
```

Ak potrebujeme explicitne zmeniť kontext nejakému objektu súborového systému, môžeme to spraviť pomocou príkazu *chcon*. Predpokladom však je, že politika nám takú zmenu povoľuje. Príkaz *chcon* je možné použiť dvoma spôsobmi:

```
chcon [-R] kontext subor

chcon [-R] [-u user] [-r role] [-t type] [-l range] subor
```

V prvom prípade je potrebné uviesť kompletný bezpečnostný kontext, v druhom prípade stačí uviesť tie zložky bezpečnostného kontextu, ktoré chceme zmeniť. V oboch prípadoch prepínač *-R* znamená rekurzívnu aplikáciu na celý podstrom *subor*. Použitie príkazu *chcon* si môžeme demonštrovať na príklade.

Najprv ako administrátor (*staff_u*) v role *sysadm_r* vytvoríme adresár a nastavíme jeho Linux-ového vlastníka na používateľa *luser*:

```
root@debian:/home/luser# id
uid=0(root) gid=0(root) groups=0(root) context=staff_u:sysadm_r:sysadm_t:s0
root@debian:/home/luser# mkdir aaa
root@debian:/home/luser# ls -ldZ aaa
drwxr-xr-x. 2 root root staff_u:object_r:user_home_t:s0 4096 Oct 14 12:45 aaa
root@debian:/home/luser# chown luser:luser aaa
```

Teraz vyskúšame ako používateľ *luser* so SELinux-ovou identitou *user_u* tento adresár použiť:

```
luser@debian:~$ cd aaa
-bash: cd: aaa: Permission denied
```

Zlyhalo to, pretože SELinux-ový používateľ tohto adresára je *staff_u* a nie *user_u* a politika vyžaduje pre bežných používateľov, aby bol SELinux-ový používateľ procesu a objektu rovnaký. Použijeme preto príkaz *chcon* na zmenu SELinux-ového používateľa na *user_u*:

```
root@debian:/home/luser# ls -ldZ aaa
drwxr-xr-x. 2 luser luser staff_u:object_r:user_home_t:s0 4096 Oct 14 12:45 aaa
root@debian:/home/luser# chcon -u user_u aaa
root@debian:/home/luser# ls -ldZ aaa
drwxr-xr-x. 2 luser luser user_u:object_r:user_home_t:s0 4096 Oct 14 12:45 aaa
```

Následne sa už používateľ *luser* k adresáru dostane:

```
luser@debian:~/aaa$ ls -laZ
total 8
drwxr-xr-x. 2 luser luser user_u:object_r:user_home_t:s0     4096 Oct 14 12:45 .
drwxr-xr-x. 3 luser luser user_u:object_r:user_home_dir_t:s0 4096 Oct 14 12:45 ..
```

Zmena kontextu príkazom *chcon* však neovplyvní informácie uložené v politike, takže pri neskoršom použití príkazu *restorecon* budú zmené kontexty opäť „opravené“ na svoje hodnoty definované v politike. Ak teda zmenu kontextu chceme zachovať do budúcnosti, môže byť vhodné upraviť informácie v politike. To je možné spraviť pomocou príkazu *semanage*:

```
semanage fcontext oper [opts] subor
```

Operácie *oper* sú rovnaké ako pri použití *semanage* na správu používateľov a mapovaní identít, voľby *opts* sú nasledovné:

- *-t type*
    - typ,

- *-s selinux_user*
  - SELinux-ový používateľ,
- *-r mls_range*
  - rozsah MLS značiek,
- *-f file_type*
  - obmedzenie druhu súboru:
    - *f* – obyčajný súbor,
    - *d* – adresár.

Parameter *subor* obsahuje regulárny výraz popisujúci cestu k objektu, ktorého bezpečnostný kontext chceme v politike definovať. Príklad použitia tohto príkazu na pridanie záznamu do politiky môže vyzerať nasledovne:

```
# semanage fcontext -a -t myapp_files_t -f d /mydata
# semanage fcontext -a -t myapp_files_t '/mydata/.*'
```

Prvý príkaz pridá záznam, ktorý definuje typ *myapp_files_t* pre adresár (*d*) */mydata*. Druhý príkaz pridá záznam, ktorý definuje typ *myapp_files_t* pre všetky objekty, ktorých cesta začína */myapp/* a pokračuje akýmikoľvek znakmi (.) v ľubovoľnom počte (*).

Na vypísanie všetkých lokálnych zmien, ktoré sme týmto príkazom do politiky vniesli, môžeme použiť nalsedujúci príkaz:

```
# semanage fcontext -l -C
SELinux fcontext    type               Context

/mydata             directory          system_u:object_r:myapp_files_t:s0
/mydata/.*          all files          system_u:object_r:myapp_files_t:s0
```

Môžeme si všimnúť, že ak neuvedieme používateľa, *semanage fcontext* použije *system_u*. Rola objektu bude týmto spôsobom vždy nastavená na *object_r*, čo je štandardná rola pre objekty.

5.5.5. Nastavenie kontextu spúšťanému programu

Pri popise pravidiel na určenie domény procesu po spustení programu sme spomenuli, že proces môže aj explicitne uviesť v akej doméne má proces po spustení programu bežať. Túto vlastnosť využíva aj príkaz *runcon*, ktorý umožňuje spustiť ľubovoľný program s určeným bezpečnostným kontextom. Samozrejme, predpokladom je, že politika umožňuje príslušný prechod z jednej domény do druhej. Príkaz *runcon* môžeme použiť dvomi spôsobmi:

```
runcon kontext príkaz

runcon [-u user] [-r role] [-t type] [-l range] príkaz
```

V prvom prípade špecifikujeme kompletný bezpečnostný kontext, v druhom prípade špecifikujeme zložky bezpečnostného kontextu, ktoré chceme zmeniť; neuvedené zostanú zachované z aktuálneho bezpečnostného kontextu. Použitie príkazu *runcon* si neskôr ukážeme na rôznych príkladoch.

5.5.6. Správa modulov

SELinux v súčasnosti umožňuje, aby politika bola modulárna, a aby sa jednotlivé moduly dali jednoduchým spôsobom do aktívnej politiky pridávať a odoberať. Na správu modulov slúži príkaz *semodule*. Môžeme ho použiť na vypísanie všetkých nainštalovaných modulov:

```
root@debian:~# semodule -l
accountsd
acct
acpi
ada
afs
aiccu
aide
aisexec
alsa
amanda
amavis
amtu
apache
apcupsd
apt
arpwatch
asterisk
auditadm
automount
…
```

Nainštalované moduly sú spojené do politiky, ktorá je nahraná do jadra OS, a ktorá je aktívne používaná. Na pridanie nového modulu do politiky potrebujeme modul v podobe binárneho balíčka (*policy package*). Tieto balíčky majú štandardne príponu *.pp* a moduly, ktoré sú súčasťou referenčnej politiky sa nachádzajú v adresári */usr/share/selinux/meno_politiky*. Balíček nainštalujeme do politiky príkazom:

```
semodule -i modul.pp
```

Pri inštalácii modulu sa overí, či je modul do politiky zakomponovateľný. Ak modul závisí na iných moduloch (lebo potrebuje, aby v politike existovali napr. ich typy, atribúty, či roly), nie je ho možné

nainštalovať do politiky, kde tieto závislosti nie sú splnené.

Modul môžeme z politiky aj odinštalovať – tým zaniknú všetky typy, atribúty a roly, ktoré modul definoval:

```
semodule -r modul
```

Ak nechceme modul z politiky odinštalovať, ale chceme ho dočasne zakázať, môžeme to spraviť príkazom:

```
semodule -d modul
```

Efekt zakázania modulu je podobný ako jeho odinštalovanie, ale balíček zostane v konfiguračnom adresári politiky a je ho možné jednoducho opäť povoliť príkazom:

```
semodule -e modul
```

Po pridaní, odobraní, zakázaní a povolení modulu dochádza k zmene aktívnej politiky. To znamená, že nám pribudnú alebo odbudnú časti politiky vrátane napr. typov a rol. Dôsledok toho je, že sa niektoré bezpečnostné kontexty môžu stať neplatné. Taktiež dochádza k zmene informácií o správnych bezpečnostných kontextoch súborov. Preto je po týchto operáciách často potrebné nanovo inicializovať kontexty súborov, najlepšie použitím príkazu *restorecon*.

5.5.7. Prepínanie módu SELinux-u

Pri popise konfigurácie SELinux-u sme spomenuli dva módy práce – *permissive* a *enforcing*. Mód práce SELinuxu je možné zistiť príkazom *getenforce* a zmeniť za behu príkazom *setenforce*. Na prechod z *enforcing* módu do *permissive* módu je, samozrejme, potrebné oprávnenie. Použitie týchto príkazov si môžeme demonštrovať na nasledujúcom príklade:

```
root@debian:~# getenforce
Enforcing
root@debian:~# setenforce 0
root@debian:~# getenforce
Permissive
root@debian:~# setenforce 1
root@debian:~# getenforce
Enforcing
```

Ako vidíme z príkladu, príkaz *setenforce* sa riadi parametrom s hodnotou 0 (*permissive*) a 1 (*enforcing*). Prepnutie do *permissive* módu efektívne znamená vypnutie všetkých kontrol, ktoré SELinux vykonáva.

SELinux tiež umožňuje prepnúť do *permissive* módu len vybrané domény. V takom prípade SELinux nebude kontrolovať operácie, ktoré sú vykonávané procesom v takejto doméne. Správa

domén v *permissive* móde sa vykonáva pomocou príkazu *semanage*:

```
semanage permissive oper doména
```

5.5.8. Ďalšie užitočné nástroje

SELinux pri zamietnutí prístupu, ak to nemá explicitne potlačené pravidlom *dontaudit*, a prípadne aj pri povolení prístupu, ak to má prikázané pravidlom *auditallow*, generuje auditné záznamy. Tie môžu byť spracované a uložené rôznym spôsobom (*auditd*, *syslog*). Jedno z miest, kde ich môžeme nájsť je aj výstup príkazu *dmesg*. V týchto záznamoch nájdeme informáciu, akú operáciu a z akého dôvodu SELinux zamietol. V nasledujúcom príklade sa pokúsime prečítať citlivý súbor */etc/shadow*, ktorý obsahuje hašovacie hodnoty hesiel používateľov:

```
root@debian:~/allowshadow# id
uid=0(root) gid=0(root) groups=0(root)
context=root:sysadm_r:sysadm_t:s0-s0:c0.c10

root@debian:~/allowshadow# dmesg -C

root@debian:~/allowshadow# cat /etc/shadow
cat: /etc/shadow: Permission denied

root@debian:~/allowshadow# dmesg
```

Napriek tomu, že sme to vykonali ako Linux-ový používateľ *root* so SELinux-ovou identitou *root* v role *sysadm_r*, čiže ako administrátor, čítanie súboru */etc/shadow* bolo zamietnuté. Ako vidíme, vo výstupe z *dmesg* sa nič neobjavilo. Dôvodom môže byť existencia *dontaudit* pravidla v politike. Našťastie SELinux umožňuje vypnúť efekt *dontaudit* pravidiel, čo je pri hľadaní problémov a ladení politiky veľmi užitočná vlastnosť. Slúži na to príkaz *semanage*:

```
semanage dontaudit off
```

Opäť zapnúť efekt *dontaudit* pravidiel sa dá príkazom:

```
semanage dontaudit on
```

Takže môžeme vypnúť *dontaudit* pravidlá a vyskúšať to ešte raz:

```
root@debian:~/allowshadow# semanage dontaudit off
root@debian:~/allowshadow# dmesg -C
root@debian:~/allowshadow# cat /etc/shadow
cat: /etc/shadow: Permission denied

root@debian:~/allowshadow# dmesg
[188606.149450] audit_printk_skb: 24 callbacks suppressed
[188606.149454] type=1400 audit(1413233421.953:3029): avc:  denied
{ read } for  pid=13562 comm="cat" name="shadow" dev=vda1 ino=131710
```

```
scontext=root:sysadm_r:sysadm_t:s0-s0:c0.c10
tcontext=system_u:object_r:shadow_t:s0 tclass=file
```

Ako teraz už vidíme z výstupu príkazu *dmesg*, SELinux odmietol udeliť právo *read* programu *cat* s bezpečnostným kontextom *root:sysadm_r:sysadm_t:s0-s0:c0.c10* na súbor *shadow* triedy *file* s bezpečnostným kontextom *system_u:object_r:shadow_t:s0*. Aby administrátor, ktorý ešte nie je zvyknutý na interpretáciu týchto záznamov, mohol rýchlejšie pochopiť dôvod zamietnutia, môže použiť príkaz *audit2why*, ktorý tieto hlásenia preloží do zrozumiteľnejšej podoby:

```
root@debian:~/allowshadow# audit2why -d
[188606.149454] type=1400 audit(1413233421.953:3029): avc:  denied
{ read } for  pid=13562 comm="cat" name="shadow" dev=vda1 ino=131710
scontext=root:sysadm_r:sysadm_t:s0-s0:c0.c10
tcontext=system_u:object_r:shadow_t:s0 tclass=file
 Was caused by:
       Missing type enforcement (TE) allow rule.

       You can use audit2allow to generate a loadable module to allow this
access.
```

Ďalším užitočným príkazom je *audit2allow*, ktorý slúži na generovanie pravidiel, ktoré by zakázanú operáciu povolili:

```
root@debian:~# audit2allow -d

#============= sysadm_t ==============
allow sysadm_t shadow_t:file read;
```

Ak usúdime, že tieto pravidlá chceme do politiky pridať, môžeme *audit2allow* nechať vygenerovať modul pre SELinux-ovú politiku, ktorý do nej budeme môcť následne pridať. Na generovanie zdrojového textu modulu slúži prepínač *-m meno_modulu*. Príkaz *audit2allow* môže vytvoriť aj binárny balíček s modulom, ktorý je možné vložiť do politiky. Na tento účel slúži prepínač *-M meno_modulu* a výsledný balíček bude uložený do súboru *meno_modulu.pp* a jeho zdrojový text do súboru *meno_modulu.te*:

```
root@debian:~/allowshadow# audit2allow -d -M allowshadow
******************** IMPORTANT ***********************
To make this policy package active, execute:

semodule -i allowshadow.pp

root@debian:~/allowshadow# cat allowshadow.te

module allowshadow 1.0;

require {
 type sysadm_t;
 type shadow_t;
```

```
 class file read;
}

#============= sysadm_t ==============
allow sysadm_t shadow_t:file read;
```

Takto vytvorený modul môžeme skúsiť vložiť do politiky:

```
root@debian:~/allowshadow# semodule -i allowshadow.pp
libsepol.check_assertion_helper: neverallow violated by allow
sysadm_t shadow_t:file { read };
libsemanage.semanage_expand_sandbox: Expand module failed
semodule:  Failed!
```

Ako vidíme, náš pokus zlyhal, pretože politika obsahuje *neverallow* pravidlo, ktoré je v konflikte s naším pravidlom, ktoré sme sa prostredníctvom modulu pokúsili do politiky pridať. Je to pekná ukážka významu *neverallow* pravidiel – autor príslušného modulu politiky usúdil, že sa jedná o natoľko citlivý súbor, že ani procesy v doméne *sysadm_t* ho nemôžu čítať. Aby sme túto bariéru prekonali, museli by sme upraviť modul referenčnej politiky, ktorý toto *neverallow* pravidlo deklaruje.

Nástroj *audit2allow* je užitočná pomôcka, ale pri jeho používaní na vytváranie modulov, ktoré povolia zakázané operácie treba postupovať obozretne. Ak sa nezamyslíme nad obsahom vygenerovaného modulu, ľahko sa nám stane, že povolíme aj operácie, ktoré s tou zamýšľanou nesúvisia, len sa stali v tom istom časovom intervale.

Druhým problémom je, že mnohé programy po zlyhaní prvej operácie nebudú pokračovať vo svojej činnosti, a teda potrebné operácie by sme týmto spôsobom zisťovali veľmi pomaly a na veľa iterácií. Tento problém sa dá čiastočne vyriešiť dočasným prepnutím SELinux-u do *permissive* módu, čím zabránime zlyhaniu operácií, ale stále získame záznamy o tom, ktoré operácie by v *enforcing* móde zlyhali. Môžeme si tento postup demonštrovať na nasledujúcom príklade, v ktorom sa pokúsime v doméne *staff_t* prečítať súbory typu *src_t*:

```
root@debian:~# setenforce 0
root@debian:~# dmesg -C
```

Teraz ako staff_t prečítame nejaký súbor v */usr/src* (sú typu *src_t*), v logu bude zaznamenaných veľa prístupov, ktoré by boli zamietnuté.

```
root@debian:~# audit2allow -d

#============= staff_t ==============
allow staff_t src_t:dir { read getattr open search };
allow staff_t src_t:file { read getattr open };
```

```
allow staff_t src_t:lnk_file { read getattr };
```

Po prezretí navrhnutých pravidiel vidíme, že vyzerajú zmysluplne a súvisia s činnosťou, ktorú chceme povoliť, takže necháme vygenerovať príslušný modul a pridáme ho do politiky (správne by sme mali po jeho vygenerované ešte overiť, že nám medzičasom nepribudli ďalšie zamietnuté prístupy, ktoré by spôsobili pridanie ďalších pravidiel – napr. kontrolou zdrojového textu modulu):

```
root@debian:~# audit2allow -d -M staff-src
******************* IMPORTANT ***********************
To make this policy package active, execute:

semodule -i staff-src.pp

root@debian:~# semodule -i staff-src.pp
root@debian:~# setenforce 1
```

Ako vidíme, modul sme úspešne nainštalovali, prepli sme SELinux do *enforcing* módu a môžeme vyskúšať nové správanie:

```
admin@debian:/usr/src/selinux-policy-src$ cat VERSION
2.20110726
```

Ako vidíme, teraz už požadované súbory čítať môžeme. Keď nový modul z politiky odstránime, dostaneme opäť pôvodné správanie, kedy ich čítať nebudeme môcť:

```
root@debian:~# semodule -r staff-src


admin@debian:/usr/src/selinux-policy-src$ cat VERSION
cat: VERSION: Permission denied
```

Pri pátraní po príčinách zamietnutia operácie môže byť užitočné vedieť vyhľadávať pravidlá a iné informácie zo SELinux-ovej politiky. Na to slúžia pomocné programy *seinfo* a *sesearch*. Program *seinfo* umožňuje zistiť informácie o typoch, atribútoch, obmedzeniach, rolách, používateľoch a ďalších prvkoch politiky:

```
root@debian:~# seinfo -u

Users: 7
   sysadm_u
   system_u
   root
   staff_u
   test_u
   user_u
   unconfined_u
```

```
root@debian:~# seinfo -r

Roles: 6
   staff_r
   user_r
   object_r
   sysadm_r
   system_r
   unconfined_r

root@debian:~# seinfo -usysadm_u -x
Users: 1
   user sysadm_u roles sysadm_r level s0 range s0-s0:c0.c1023;
```

Pomocou prepínačov *-u* (používatelia), *-r* (roly), *-a* (atribúty), *-t* (typy) môžeme špecifikovať, o aké informácie máme záujem. Ak za prepínačom uvedieme identifikátor (bez medzery), dozvieme sa údaje len o príslušnom prvku. Prepínačom *-x* si vyžiadame rozšírené informácie – napr. povolené roly používateľa, atribúty typu, alebo typy, ktoré majú atribút.

Program *sesearch* umožňuje vyhľadávať v politike pravidlá podľa zdrojového (*-s*) a cieľového kontextu (*-t*), triedy objektu (*-c*) a práv (*-p*). Druh pravidla, ktoré chceme hľadať špecifikujeme prvým parametrom príkazu nasledovne:

- *-A* alebo *--allow* – pravidlo *allow*,
- *--neverallow* – pravidlo *neverallow*,
- *--auditallow* – pravidlo *auditallow*,
- *--dontaudit* – pravidlo *dontaudit*,
- *-T* alebo *--type* – pravidlá *type_transition* alebo *type_change*,
- *--role_allow* – pravidlo *allow* pre roly,
- *--role_trans* – pravidlo *role_transition*,
- *--range_trans* – pravidlo *range_transition*.

Ak napríklad chceme nájsť všetky pravidlá *allow*, kde je zdrojovou doménou *user_t*, triedou *process* a právom *transition*, čiže všetky pravidlá, ktoré bežnému používateľovi umožňujú spustiť program v inej doméne, môžeme použiť príkaz:

```
root@debian:~# sesearch -A -s user_t -c process -p transition
allow user_t bluetooth_helper_t:process { getattr ptrace sigchld sigkill signal signull
sigstop transition };
allow user_t cdrecord_t:process { getattr ptrace sigchld sigkill signal signull sigstop
```

```
transition };
allow user_t chfn_t:process transition;
allow user_t chkpwd_t:process { getattr transition };
allow user_t chromium_t:process { getattr sigchld sigkill signal signull sigstop transition };
allow user_t dirmngr_t:process { getattr ptrace sigchld sigkill signal signull sigstop
transition };
allow user_t evolution_alarm_t:process { getattr noatsecure ptrace sigchld sigkill signal
signull sigstop transition };
allow user_t evolution_exchange_t:process { getattr noatsecure ptrace sigchld sigkill signal
signull sigstop transition };
allow user_t evolution_server_t:process { getattr noatsecure ptrace sigchld sigkill signal
signull sigstop transition };
allow user_t evolution_t:process { getattr noatsecure ptrace sigchld sigkill signal signull
sigstop transition };
allow user_t evolution_webcal_t:process { getattr noatsecure ptrace sigchld sigkill signal
signull sigstop transition };
allow user_t exim_t:process transition;
   …
```

5.5.9. Inštalácia SELinux-u

Na podporu SELinux-u v OS Linux je potrebných niekoľko súčastí. Potrebná podpora v jadre systému je dnes štandardnou súčasťou jadra pre všetky známejšie distribúcie. Okrem podpory v jadre systému je však potrebná podpora v rôznych pomocných programoch a tiež špecifické programy slúžiace na správu SELinux-u. Známejšie distribúcie OS Linux dnes štandardne obsahujú štandardné programy so zahrnutou podporou pre SELinux. Špecifické programy pre SELinux sú zvyčajne súčasťou príslušných balíkov danej distribúcie. Ukážeme si inštaláciu SELinux-u na príklade distribúcie Debian 10, pre iné distribúcie Linux-u bude obdobná.

Prvým krokom je inštalácia balíkov programov na podporu SELinux-u:

- *selinux-basics*
    - prostredníctvom závislostí spôsobí inštaláciu ďalších balíkov, ako najmä *checkpolicy*, *policycoreutils* a *selinux-utils*,
- *newrole*
    - obsahuje príkaz *newrole* na zmenu roly,
- *selinux-policy-default*
    - obsahuje referenčnú politiku,
- *selinux-policy-dev*
    - je užitočný pre vývoj vlastných modulov, inak nie je potrebný.

Pri inštalácii referenčnej politiky inštalačný skript analyzuje nainštalované balíky a automaticky do politiky pridá príslušné moduly.

Následne potrebujeme upraviť konfiguráciu *boot-loader*-a a subsystému PAM. To za nás spraví skript *selinux-activate* z balíčka *selinux-basics*. V prípade *boot-loader*-a ide predovšetkým o nastavenie parametra jadra *security=selinux*. Po reštarte systému sa podľa politiky inicializujú bezpečnostné kontexty objektov súborového systému a systém je pripravený na prevádzku.

Keď do systému nainštalujeme nový subsystém, je vhodné zistiť, či k nemu existuje príslušný modul do politiky a pomocou *semodule* ho pridať a pomocou *restorecon* reinicializovať bezpečnostné kontexty súborov.

5.6. SELinux MCS

V tejto časti si priblížime variant MLS nazývaný *Multi-Category Security* (*MCS*) v podaní referenčnej politiky SELinux-u. Model MCS nevyužíva prvú zložku bezpečnostnej značky (formálne definuje len jednu úroveň), ale využíva množinovú zložku bezpečnostnej značky. Objekty majú dolnú a hornú bezpečnostnú značku rovnakú, a táto vyjadruje oblasti, ktorých sa údaje uložené v objekte týkajú (môžu týkať). Horná bezpečnostná značka vyjadruje množinu oblastí, s ktorými je subjekt oprávnený prichádzať do styku a slúži na zabezpečenie toho, aby subjekt nemohol manipulovať s údajmi, ktoré sa týkajú oblasti, na ktorú subjekt nie je oprávnený. Na to, aby subjekt nemohol (napr. v dôsledku zneužitia chyby) sprístupniť citlivé údaje týkajúce sa niektorej oblasti príliš nedôveryhodným subjektom, slúži dolná bezpečnostná značka subjektu, ktorá definuje minimálnu úroveň ochrany objektov, do ktorých môže subjekt zapisovať.

Ak má subjekt dolnú bezpečnostnú značku L, hornú H a objekt bezpečnostnú značku O, tak:

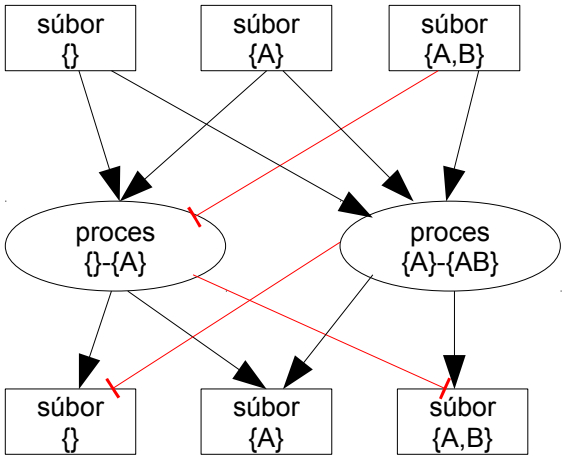
- subjekt môže čítať z objektu, ak $H \geq O$,
- subjekt môže zapisovať do objektu, meniť atribúty objektu, premenovať alebo vymazať objekt, ak $H \geq O$ a zároveň $L \leq O$,
- subjekt môže vytvoriť nový objekt v adresári so značkou D, ak $H \geq D$.

Tieto pravidlá sú slabšie ako napr. pravidlá modelu Bell – La Padula, pretože celkom nebránia prenosu informácií z objektu s vyššou značkou do objektu s nižšou značkou.

Takáto MCS politika sa dá využiť v praxi napríklad na oddelenie údajov rôznej agendy (napr. osobné údaje, finančné údaje, ...) a riadenie prístupu k nim na báze toho, či používateľ alebo proces je alebo nie je oprávnený s údajmi danej agendy pracovať. Ak budú mať súbory s osobnými údajmi nastavenú kategóriu „osobné údaje“, tak s nimi nebude môcť pracovať žiadny proces, ktorý nemá vo svojej hornej značke kategóriu „osobné údaje“. Ak mu navyše nastavíme kategóriu „osobné údaje“ aj do dolnej značky, nebude môcť nič zapísať do súborov, ktoré nie sú chránené ako osobné

údaje. Týmto spôsobom môžeme zabrániť úniku osobných údajov zo systému.

Pre ilustráciu možných tokov informácií medzi subjektami a objektami uvažujme dve kategórie – *A* a *B*. Nasledujúci obrázok ilustruje možné toky:



Čierne šípky označujú povolené smery prenosu, červené čiary zakončené zarážkou označujú zakázané smery prenosu.

Fungovanie týchto obmedzení si môžeme demonštrovať na nasledujúcom príklade.

Na začiatku majme dva súbory *a* a *b*. Zmeníme bezpečnostnú značku súboru *a* na *s0:c0*:

```
admin@debian:~/mcstest$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:staff_t:s0-s0:c0.c10
admin@debian:~/mcstest$ chcon -l s0:c0 a
admin@debian:~/mcstest$ ls -lZ
total 8
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0:c0 2 Oct 14 16:04 a
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0    2 Oct 14 16:04 b
```

Teraz si zúžime rozsah bezpečnostných značiek na *s0*, teda bez kategórií. Na to použijeme príkaz *runcon* a spustíme si s novým kontextom *shell* a skúsime prečítať a zmeniť obsah súborov *a* a *b*:

```
admin@debian:~/mcstest$ runcon -l s0 bash
admin@debian:~/mcstest$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:staff_t:s0
admin@debian:~/mcstest$ cat a
cat: a: Permission denied
admin@debian:~/mcstest$ cat b
B
admin@debian:~/mcstest$ echo a >> a
bash: a: Permission denied
admin@debian:~/mcstest$ echo a >> b
```

Vidíme, že zatiaľ čo so súborom *b* (bez kategórií) pracovať môžeme, so súborom *a* nie, pretože nemáme kategóriu *c0*. Ak by sme sa pokúsili rozšíriť si rozsah MLS značiek, tak táto operácia zlyhá, pretože MCS politka prirodzene neumožňuje rozsah značiek rozširovať – len zužovať:

```
admin@debian:~/mcstest$ runcon -l s0:c0 bash
runcon: bash: Permission denied
admin@debian:~/mcstest$ exit
```

Ďalším experimentom bude spustenie *shell*-u s hornou aj dolnou značkou s kategóriou *c0*:

```
admin@debian:~/mcstest$ runcon -l s0:c0 bash
admin@debian:~/mcstest$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:staff_t:s0:c0
admin@debian:~/mcstest$ cat a
A
admin@debian:~/mcstest$ cat b
B
a
admin@debian:~/mcstest$ echo a >> a
admin@debian:~/mcstest$ echo a >> b
bash: b: Permission denied
admin@debian:~/mcstest$ echo a >> c
admin@debian:~/mcstest$ ls -lZ
total 12
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0:c0 4 Oct 14 16:06 a
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0    4 Oct 14 16:05 b
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0:c0 2 Oct 14 16:09 c
admin@debian:~/mcstest$ rm b
rm: remove write-protected regular file `b'? y
rm: cannot remove `b': Permission denied
```

V tomto prípade vidíme, že vieme čítať oba súbory, ale do súboru bez kategórie (*b*) nevieme zapisovať. Keď vytvoríme nový súbor (*c*), jeho bezpečnostná značka bude inicializovaná na našu dolnú značku, v tomto prípade obsahujúcu kategóriu *c0*.

Pravidlá pre čítanie a zápis do súborov sú definované v politike pomocou *mlsconstrain* pravidiel. Aktuálna politika v Debian 10, žiaľ, oproti minulosti odstránila obmedzenie požadujúce dominanciu bezpečnostnej značky objektu nad dolnou hranicou rozsahu subjektu, takže umožňuje zapisovať do súborov s bezpečnostnou značkou, ktorej dominuje horná hranica rozsahu subjektu (teda rovnako ako pri čítaní). Navyše táto nová verzia politiky odstránila povolenie *setexec* pre bežných používateľov, čo im znemožňuje použitie príkazu *runcon* na zmenu rozsahu. Stále je však aj takto pozmenená politika užitočná na označovanie súborov patriacich do rôznych oblastí príslušnými kategóriami a prideľovanie príslušných kategórií používateľom, ktorí majú mať oprávnenie so súbormi danej kategórie pracovať. Samozrejme, je možné si politiku aj upraviť podľa špecifických potrieb.

Aby bol model MCS použiteľnejší aj v bežnom prostredí, umožňuje nám SELinux pomenovať jednotlivé bezpečnostné značky zrozumiteľnejšími menami, ktoré sa potom budú vo výstupoch používať namiesto notácie tvaru *s0:c0*, a pod. Na tento účel slúži služba *mcstransd*, ktorá zabezpečuje preklad medzi SELinux-ovou MLS značkou a zrozumiteľnejším menom. Služba je súčasťou balíka *mcstrans*, ktorý, žiaľ, v Debian 10 nie je kompatibilný s distribuovanou verziou politiky. Konfigurácia mien sa nachádza v súbore

```
/etc/selinux/meno_politiky/setrans.conf
```

a má jednoduchý formát:

```
s0:c0=sukromne
s0:c1=pracovne
```

Po zmene tohto konfiguračného súboru je potrebné službu *mcstransd* reštartovať, a potom môžeme vyskúšať preklad mien v praxi:

```
admin@debian:~/mcstest$ ls -lZ
total 12
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:sukromne 4 Oct 14 16:06 a
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:s0       4 Oct 14 16:05 b
-rw-r--r--. 1 admin admin staff_u:object_r:user_home_t:sukromne 2 Oct 14 16:09 c
```

Ako môžeme vidieť, bezpečnostné značky, ktoré nemajú priradené mená sa zobrazujú štandardne a bezpečnostné značky s priradenými menami sa zobrazujú zrozumiteľnejšie.

Na ďalšie uľahčenie práce s úpravou kategórií súborov slúži príkaz *chcat*. Ak chceme kategóriu *cat* súboru nastaviť bez ohľadu na predchádzajúce nastavenie, môžeme použiť:

```
chcat cat subor
```

Na pridanie kategórie *cat* k existujúcim môžeme použiť:

```
chcat +cat subor
```

Na odobratie kategórie *cat* od existujúcich môžeme použiť:

```
chcat -- -cat subor
```

A na odstránenie všetkých kategórií môžeme použiť príkaz:

```
chcat -d subor
```

Kategórie používateľa (jeho hornú hranicu) môžeme nastaviť príkazom:

```
chcat -l cat selinux_user
```

alebo pomocou *semanage login*.

Pri používaní príkazu *chcat* môžeme používať kategórie pomenované SELinux-ovými identifikátormi kategórií z politiky alebo zrozumiteľnejšími menami definovanými pomocou služby *mcstransd*.

5.7. Príklad vytvorenia vlastného modulu politiky SELinux-u

Na záver kapitoly venovanej SELinux-u sa pozrieme na jednoduchý príklad vytvorenia vlastného modulu politiky pre veľmi jednoduchú aplikáciu. Naším cieľom bude vytvorenie modulu *myapp*,

ktorý vytvorí novú doménu tak, že:

- bude v nej možné spúšťať bežné programy,
- bežní používatelia budú môcť spustiť programy v tejto doméne,
- nebude mať prístup k používateľským dátam,
- bude mať prístup k vlastným dátam v */mydata*,
- bežní používatelia nebudú mať prístup k */mydata*.

Zdrojový text modulu SELinux-ovej politiky sa skladá z troch súborov:

- *modul.te*
  - obsahuje definície vlastných typov, atribútov, rol, pravidiel, môže používať rozhrania (*interface*) iných modulov,
- *modul.if*
  - definuje rozhrania (*interface*) modulu, prostredníctvom ktorých môžu iné moduly využívať jeho služby,
- *modul.fc*
  - definuje pravidlá pre priradenie bezpečnostných kontextov súborom a adresárom.

Zdrojový text modulu je následne potrebné skompilovať do binárneho tvaru a spojiť do výsledného binárneho balíčka (*policy package*) – *modul.pp*. Na tento proces je potrebných niekoľko nástrojov, avšak detaily nie sú až také dôležité, pretože všetko pre nás zastreší štandardný program *make*, ak mu dáme poskytneme správny súbor *Makefile*. Ten si do adresára s našimi zdrojovými textami modulu skopírujeme z

```
/usr/share/doc/selinux-policy-dev/examples/Makefile
```

Keď spustíme príkaz *make* v adresári s týmto súborom, nájde v tomto adresári všetky súbory s príponou *.te* a vytvorí z nich výsledné binárne balíčky s príponou *.pp*. Zvyčajne je vhodné, aby sme v adresári mali iba jeden modul.

Môžeme teda prikročiť k vytvoreniu prvej verzie nášho modulu, ktorý budeme postupne dopĺňať:

myapp.te

```
policy_module(myapp, 1.0.0)
type myapp_t;
```

```
type myapp_exec_t;
type myapp_files_t;

# povolime zakladne nalezitosti aplikacnej domeny
# myapp_t oznaci ako aplikacnu domenu (napr. jej povoli pouzivat
# kniznice)
# myapp_exec_t oznaci ako jej "entrypoint" a spravi ho spustitelny

application_domain(myapp_t, myapp_exec_t)
```

Každý modul musí mať svoj názov a číslo verzie – tieto údaje sa deklarujú makrom *policy_module*. V našom module deklarujeme tri typy:

- *myapp_t* – naša doména,
- *myapp_exec_t* – typ spustiteľných súborov, ktorých spustením sa bude prechádzať do našej domény,
- *myapp_files_t* – typ pre súbory, ku ktorým bude mať naša doména prístup.

Rozhranie *application_domain* je rozhranie modulu *application*, ktoré nášmu typu *myapp_t* priradí atribút označujúci všetky aplikačné domény, typu *myapp_exec_t* priradí potrebné atribúty, ktoré prostredníctvom pravidiel v rôznych moduloch zabezpečia, že súbory s týmto typom budú spúšťateľné používateľmi, a zabezpečí, že programy typu *myapp_exec_t* budú môcť byť vstupným bodom do domény *myapp_t*.

Prvú verziu nášho modulu môžeme skúsiť skompilovať a nainštalovať do politiky:

```
root@debian10:/home/admin/myapp# make
Compiling default myapp module
...
Creating default myapp.pp policy package
/usr/bin/semodule_package -o myapp.pp -m tmp/myapp.mod -f tmp/myapp.mod.fc
rm tmp/myapp.mod tmp/myapp.mod.fc

root@debian10:/home/admin/myapp# semodule -i myapp.pp
```

Vidíme, že kompilácia aj inštalácia modulu prebehli bez problémov, takže môžeme prikročiť k prvým testom. V adresári */usr/local/bin* máme pripravené tri programy:

- *hw* – je triviálny program, ktorý na výstup vypíše text „Hello World“ a skončí,
- *mybash* – je kópia *shell*-u, ktorú budeme chcieť spúšťať v našej doméne,
- *myid* – je kópia programu *id*.

Keďže sme úspešne nainštalovali náš modul, naše nové typy pribudli do politiky. Takže týmto programom môžeme nastaviť typ na *myapp_exec_t* a môžeme ich skúsiť spustiť:

```
root@debian10:/usr/local/bin# ls -lZ
total 964
-rwxr-xr-x. 1 root staff system_u:object_r:bin_t:s0   4862 Oct 14 19:14 hw
-rwxr-xr-x. 1 root staff system_u:object_r:bin_t:s0 941252 Oct 15 18:07 mybash
-rwxr-xr-x. 1 root staff system_u:object_r:bin_t:s0  34396 Oct 14 18:36 myid

root@debian10:/usr/local/bin# chcon -t myapp_exec_t *

admin@debian10:~$ ls -lZ /usr/local/bin/
total 1208
-rwxr-xr-x. 1 root root system_u:object_r:myapp_exec_t:s0   16600 Apr  5 03:18 hw
-rwxr-xr-x. 1 root root system_u:object_r:myapp_exec_t:s0 1168776 Apr  5 03:17 mybash
-rwxr-xr-x. 1 root root system_u:object_r:myapp_exec_t:s0   43808 Apr  5 03:17 myid

admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin) context=staff_u:staff_r:staff_t:s0-
s0:c0.c10
admin@debian10:~$ myid
uid=1002(admin) gid=1002(admin) groups=1002(admin) context=staff_u:staff_r:staff_t:s0-
s0:c0.c10
admin@debian10:~$ hw
Hello world
```

Ako vidíme, programy sa spustiť dajú, fungujú, no zatiaľ bežia v doméne, z ktorej boli spustené. Ďalším krokom, ktorý môžeme vyskúšať, je spustenie explicitne v našej doméne pomocou príkazu *runcon*:

```
admin@debian10:~$ runcon -t myapp_t hw
runcon: invalid context: 'staff_u:staff_r:myapp_t:s0-s0:c0.c10': Invalid
```

Ako však vidíme, to sa nám nepodarilo, pretože požadovaný kontext nie je platný. Dôvodom neplatnosti je to, že rola *staff_r* nemá priradenú povolenú doménu *myapp_t*.

Doplníme preto náš modul o povolenie domény *myapp_t* pre rolu *staff_r*. Rola *staff_r*, však nie je definovaná v našom module, takže si ju potrebujeme naimportovať zo zvyšku politiky pomocou konštrukcie *require { ... }*. Už skôr sme spomenuli, že používanie typov a iných prvkov definovaných v iných moduloch nie je v referenčnej politike korektné. Technicky to však možné je, a zatiaľ túto nekorektnosť môžeme ignorovať.

Nová verzia nášho modulu bude vyzerať nasledovne (zvýraznené sú nové časti):

myapp.te

```
policy_module(myapp, 1.0.1)

type myapp_t;
type myapp_exec_t;
type myapp_files_t;

# povolime zakladne nalezitosti aplikacnej domeny
# myapp_t oznaci ako aplikacnu domenu (napr. jej povoli pouzivat kniznice)
# myapp_exec_t oznaci ako jej "entrypoint" a spravi ho spustitelny

application_domain(myapp_t, myapp_exec_t)

require{
  role staff_r;
}

role staff_r types myapp_t;
```

Môžeme ju skompilovať a nainštalovať do politiky a následne vyskúšať:

```
root@debian10:/home/admin/myapp# make
root@debian10:/home/admin/myapp# semodule -i myapp.pp

admin@debian10:~$ runcon -t myapp_t hw
runcon: unable to set security context
'staff_u:staff_r:myapp_t:s0-s0:c0.c10': Permission denied

root@debian10:/home/admin/myapp# dmesg
[51362.686284] audit: type=1400 audit(1617627364.381:4295):
avc:  denied  { setexec } for  pid=3476 comm="runcon"
scontext=staff_u:staff_r:staff_t:s0-s0:c0.c10
tcontext=staff_u:staff_r:staff_t:s0-s0:c0.c10 tclass=process
permissive=0
```

Tento raz bol už požadovaný kontext platný, ale spustenie v našej doméne bolo aj tak zamietnuté. Ako vidíme z auditného záznamu, dôvodom je chýbajúce pravidlo povoľujúce operáciu *setexec*, ktorú proces potrebuje na nastavenie bezpečnostného kontextu (o to sa príkaz *runcon* pokúsil). Ako sme už spomenuli v predchádzajúcich častiach, nová politika toto oprávnenie už bežným používateľom nedáva. Môžeme však pokus zopakovať s rolou *sysadm_r* (po pridaní *myapp_t* medzi povolené domény tejto roly):

```
admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:sysadm_r:sysadm_t:SystemLow-s0:c0.c10
admin@debian10:~$ runcon -t myapp_t hw
runcon: 'hw': Permission denied

root@debian10:/home/admin/myapp# dmesg
[51855.239165] audit: type=1400 audit(1617627856.947:4353):
avc:  denied  { transition } for  pid=3554 comm="runcon"
path="/usr/local/bin/hw" dev="sda2" ino=408787
scontext=staff_u:sysadm_r:sysadm_t:s0-s0:c0.c10
tcontext=staff_u:sysadm_r:myapp_t:s0-s0:c0.c10 tclass=process
permissive=0
```

Ako vidíme, spustenie programu v našej doméne bolo opäť zamietnuté a z auditného záznamu vidieť, že príčinou je chýbajúce oprávnenie *transistion* povoľujúce prechod z domény *sysadm_t* do domény *myapp_t*.

Na pridanie tohto pravidla využijeme jedno zo štandardných makier používaných v referenčnej politike zo súboru

```
/usr/share/selinux/devel/include/support/misc_patterns.spt/misc_patterns.spt
```

Tento súbor definuje makro *spec_domtrans_pattern*, ktoré má tri parametre – zdrojovú doménu, typ

spustiteľného súboru a cieľovú doménu. Definícia tohto makra je nasledovná:

```
define(`spec_domtrans_pattern',`
        allow $1 self:process setexec;
        domain_transition_pattern($1,$2,$3)

        allow $3 $1:fd use;
        allow $3 $1:fifo_file rw_fifo_file_perms;
        allow $3 $1:process sigchld;
')

define(`domain_transition_pattern',`
        allow $1 $2:file mmap_exec_file_perms;
        allow $1 $3:process transition;
        dontaudit $1 $3:process { noatsecure siginh rlimitinh };
')
```

Ako vidíme, toto makro (pomocou ďalšieho) pridá právo *transition* medzi doménami, a tiež pridá chýbajúce právo *setexec*. Takže môžeme upraviť náš modul:

myapp.te

```
policy_module(myapp, 1.0.2)

type myapp_t;
type myapp_exec_t;
type myapp_files_t;

application_domain(myapp_t, myapp_exec_t)

require{
  role staff_r;
  type staff_t;
}

role staff_r types myapp_t;
spec_domtrans_pattern(staff_t,myapp_exec_t,myapp_t)
```

Po skompilovaní a nainštalovaní novej verzie do politiky ju môžeme vyskúšať:

```
admin@debian10:~$ runcon -t myapp_t hw
admin@debian10:~$

root@debian10:/home/admin/myapp# dmesg
[52589.937175] audit: type=1400 audit(1617628591.654:4390): avc:  denied  {
read write } for  pid=3696 comm="hw" path="/dev/pts/1" dev="devpts" ino=4
scontext=staff_u:staff_r:myapp_t:s0-s0:c0.c10
tcontext=staff_u:object_r:user_devpts_t:s0 tclass=chr_file permissive=0

[52589.937182] audit: type=1400 audit(1617628591.654:4391): avc:  denied  {
use } for  pid=3696 comm="hw" path="/dev/pts/1" dev="devpts" ino=4
scontext=staff_u:staff_r:myapp_t:s0-s0:c0.c10
tcontext=system_u:system_r:sshd_t:s0-s0:c0.c1023 tclass=fd permissive=0
```

Ako vidíme, spustenie programu *hw* sa už podarilo, ale nič nevypísal. Keď sa pozrieme na auditné záznamy, tak zistíme, že mu boli odopreté práva *read* a *write* na znakové zariadenie typu

user_devpts_t a právo na použitie popisovača súborov (*file descriptor*) typu *sshd_t*. Tieto chyby súvisia s použitím pseudoterminálového zariadenia na štandardný vstup a výstup. Takže náš program síce úspešne pracoval, ale nemohol komunikovať. Vidíme, že SELinux naozaj kontroluje aj operácie so zdedenými popisovačmi súborov. Tieto problémy môžeme ľahko vyriešiť použitím rozhraní *domain_use_interactive_fds* a *userdom_use_inherited_user_terminals*. Takže opäť upravíme náš modul:

myapp.te

```
policy_module(myapp, 1.0.3)
type myapp_t;
type myapp_exec_t;
type myapp_files_t;
application_domain(myapp_t, myapp_exec_t)

# povolime pristup na zdedene fd (vstup/vystup)
domain_use_interactive_fds(myapp_t)

# povolime pristup na terminal pouzivatela
userdom_use_inherited_user_terminals(myapp_t)

require{
  role staff_r;
  type staff_t;
}

role staff_r types myapp_t;
spec_domtrans_pattern(staff_t,myapp_exec_t,myapp_t)
```

Po skompilovaní a inštalácii môžeme opäť testovať:

```
admin@debian10:~$ runcon -t myapp_t hw
Hello world
admin@debian10:~$ runcon -t myapp_t myid
uid=1002 gid=1002 groups=1002
```

Ako môžeme vidieť, triviálny program *hw* už teraz funguje, no *myid* celkom nie – nedokáže zistiť mená Linux-ového používateľa a skupiny ani bezpečnostný kontext. Aby sme tieto problémy odstránili, povolíme našej doméne prostredníctvom niekoľkých rozhraní iných modulov zopár ďalších operácií. Do modulu pridáme riadky:

```
# povolime citanie beznych suborov z /etc
files_read_etc_files(myapp_t)

# povolime pristup potebny na zistenie, ci je aktivny SELinux
selinux_get_fs_mount(myapp_t)

# povolime citanie konfiguracie SELinuxu
seutil_read_config(myapp_t)

# povolime pristup k suborom pre lokalizaciu prostredia
miscfiles_read_localization(myapp_t)
```

```
# povolime si spustanie zakladnych programov v /bin a pod.
corecmd_exec_bin(myapp_t)

# shell bude potrebovat setpgid na nastavenie vytvorenie procgroup

allow myapp_t self:process setpgid;
```

Následne modul znovu skompilujeme a nainštalujeme a môžeme ho vyskúšať:

```
admin@debian10:~$ runcon -t myapp_t hw
Hello world
admin@debian10:~$ runcon -t myapp_t myid
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:myapp_t:s0-s0:c0.c10
admin@debian10:~$ runcon -t myapp_t mybash
mybash: /home/admin/.bashrc: Permission denied
admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:myapp_t:s0-s0:c0.c10
admin@debian10:~$ ls
ls: cannot open directory '.': Permission denied
```

Ako vidíme, už funguje nie len *hw*, ale aj *myid* a podarilo sa spustiť aj *mybash* a z neho štandardný príkaz *id*, ktorý jasne preukazuje, že úspešne beží v našej doméne. Chybové hlášky o zamietnutí prístupu k *.bashrc* ako aj o zamietnutí čítania domovského adresára tiež signalizujú úspech, pretože sme úmyselne chceli, aby naša doména nemala prístup k používateľským údajom, ktorých je *.bashrc* súčasťou.

Ešte kým sme v *shell*-i bežiacom v našej novej doméne, môžeme vyskúšať prístup k niektorým ďalším objektom:

```
admin@debian10:~$ ls -l /
ls: cannot access '/run': Permission denied
ls: cannot access '/boot': Permission denied
ls: cannot access '/initrd.img.old': Permission denied
ls: cannot access '/var': Permission denied
ls: cannot access '/vmlinuz.old': Permission denied
ls: cannot access '/srv': Permission denied
ls: cannot access '/lost+found': Permission denied
ls: cannot access '/mnt': Permission denied
ls: cannot access '/initrd.img': Permission denied
ls: cannot access '/root': Permission denied
ls: cannot access '/tmp': Permission denied
ls: cannot access '/media': Permission denied
ls: cannot access '/vmlinuz': Permission denied
ls: cannot access '/home': Permission denied
total 12
lrwxrwxrwx.   1 root root    7 Apr  4 12:13 bin -> usr/bin
d??????????   ? ?    ?       ?            ? boot
drwxr-xr-x.  16 root root 3120 Apr  5 00:40 dev
drwxr-xr-x.  75 root root 4096 Apr  5 14:10 etc
d??????????   ? ?    ?       ?            ? home
l??????????   ? ?    ?       ?            ? initrd.img
l??????????   ? ?    ?       ?            ? initrd.img.old
lrwxrwxrwx.   1 root root    7 Apr  4 12:13 lib -> usr/lib
```

```
lrwxrwxrwx.   1 root root    9 Apr  4 12:13 lib32 -> usr/lib32
lrwxrwxrwx.   1 root root    9 Apr  4 12:13 lib64 -> usr/lib64
lrwxrwxrwx.   1 root root   10 Apr  4 12:13 libx32 -> usr/libx32
d??????????   ? ?    ?       ?            ? lost+found
d??????????   ? ?    ?       ?            ? media
d??????????   ? ?    ?       ?            ? mnt
drwxr-xr-x.   2 root root 4096 Apr  4 12:13 opt
dr-xr-xr-x. 126 root root    0 Apr  5 00:40 proc
d??????????   ? ?    ?       ?            ? root
d??????????   ? ?    ?       ?            ? run
lrwxrwxrwx.   1 root root    8 Apr  4 12:13 sbin -> usr/sbin
d??????????   ? ?    ?       ?            ? srv
dr-xr-xr-x.  13 root root    0 Apr  5 00:40 sys
d??????????   ? ?    ?       ?            ? tmp
drwxr-xr-x.  13 root root 4096 Apr  4 12:13 usr
d??????????   ? ?    ?       ?            ? var
l??????????   ? ?    ?       ?            ? vmlinuz
l??????????   ? ?    ?       ?            ? vmlinuz.old
```

Ako môžeme vidieť, prístupných máme len niekoľko systémových adresárov, ktoré obsahujú súbory, ktoré obsahujú súbory nevyhnutné na beh programov, iné adresáre sú pre nás neprístupné.

Ďalším krokom vo vývoji nášho modulu bude pridanie pravidiel na prácu s našimi vlastnými údajmi:

myapp.te

```
policy_module(myapp, 1.0.5)
type myapp_t;
type myapp_exec_t;
type myapp_files_t;
application_domain(myapp_t, myapp_exec_t)
domain_use_interactive_fds(myapp_t)
userdom_use_inherited_user_terminals(myapp_t)
files_read_etc_files(myapp_t)
selinux_get_fs_mount(myapp_t)
seutil_read_config(myapp_t)
miscfiles_read_localization(myapp_t)
corecmd_exec_bin(myapp_t)
allow myapp_t self:process setpgid;

files_type(myapp_files_t)
allow myapp_t myapp_files_t:dir *;
allow myapp_t myapp_files_t:file *;

require{
  role staff_r;
  type staff_t;
}

role staff_r types myapp_t;
spec_domtrans_pattern(staff_t,myapp_exec_t,myapp_t)
```

Rozhranie *files_type* priradí vhodné atribúty nášmu typu *myapp_files_t*, aby mohol byť použitý ako typ pre súbory (napr. umožní, aby takéto súbory mohli byť vytvorené v súborových systémoch).

Následne vytvoríme adresár */mydata* a nastavíme mu typ *myapp_files_t*:

```
root@debian10:/home/admin/myapp# mkdir /mydata
root@debian10:/home/admin/myapp# chcon -t myapp_files_t /mydata
root@debian10:/home/admin/myapp# chown admin /mydata
root@debian10:/home/admin/myapp# ls -ldZ /mydata
drwxr-xr-x. 2 admin root staff_u:object_r:myapp_files_t:SystemLow 4096 Apr  5 15:53 /mydata
```

a môžeme vyskúšať správanie:

```
admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:staff_t:s0-s0:c0.c10
admin@debian10:~$ ls -la /mydata
ls: cannot access '/mydata': Permission denied
admin@debian10:~$ runcon -t myapp_t mybash
mybash: /home/admin/.bashrc: Permission denied
admin@debian10:~$ ls -la /mydata/
total 8
drwxr-xr-x.  2 admin root 4096 Apr  5 15:53 .
drwxr-xr-x. 19 root  root 4096 Apr  5 15:53 ..
admin@debian10:~$ echo yes > /mydata/file
admin@debian10:~$ ls -laZ /mydata/
total 12
drwxr-xr-x.  2 admin root  staff_u:object_r:myapp_files_t:s0 4096 Apr  5 15:57 .
drwxr-xr-x. 19 root  root  system_u:object_r:root_t:s0       4096 Apr  5 15:53 ..
-rw-r--r--.  1 admin admin staff_u:object_r:myapp_files_t:s0    4 Apr  5 15:57 file
```

Ako môžeme vidieť, z bežnej používateľskej domény (*staff_t*) je adresár */mydata* neprístupný, no z našej domény (*myapp_t*) prístupný je.

Zatiaľ sme na spustenie programu v našej doméne používali príkaz *runcon*. Ďalším krokom bude úprava modulu tak, aby k zmene domény dochádzalo automaticky. Opäť na to využijeme štandardné makro *domtrans_pattern* definované nasledovne:

```
define(`domtrans_pattern',`
        domain_auto_transition_pattern($1,$2,$3)

        allow $3 $1:fd use;
        allow $3 $1:fifo_file rw_inherited_fifo_file_perms;
        allow $3 $1:process sigchld;
')

define(`domain_auto_transition_pattern',`
        domain_transition_pattern($1,$2,$3)
        type_transition $1 $2:process $3;
')
```

Toto makro, na rozdiel od *spec_domtrans_pattern*, zabezpečuje pomocou pravidla *type_transition* automatický prechod medzi doménami pri spustení programu príslušného typu. Náš modul teda bude vyzerať nasledovne:

myapp.te

```
policy_module(myapp, 1.0.6)
type myapp_t;
type myapp_exec_t;
type myapp_files_t;
application_domain(myapp_t, myapp_exec_t)
domain_use_interactive_fds(myapp_t)
userdom_use_inherited_user_terminals(myapp_t)
files_read_etc_files(myapp_t)
selinux_get_fs_mount(myapp_t)
seutil_read_config(myapp_t)
miscfiles_read_localization(myapp_t)
corecmd_exec_bin(myapp_t)
allow myapp_t self:process setpgid;
files_type(myapp_files_t)
allow myapp_t myapp_files_t:dir *;
allow myapp_t myapp_files_t:file *;


require{
  role staff_r;
  type staff_t;
}

role staff_r types myapp_t;
domtrans_pattern(staff_t,myapp_exec_t,myapp_t)
```

Keď nainštalujeme novú verziu do politiky, môžeme ju opäť vyskúšať:

```
admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:staff_t:s0-s0:c0.c10
admin@debian10:~$ myid
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:myapp_t:s0-s0:c0.c10
admin@debian10:~$ mybash
mybash: /home/admin/.bashrc: Permission denied
admin@debian10:~$ id
uid=1002(admin) gid=1002(admin) groups=1002(admin)
context=staff_u:staff_r:myapp_t:s0-s0:c0.c10
admin@debian10:~$ exit
```

Teraz už všetko funguje automaticky bez použitia *runcon*.

Ako sme spomenuli na začiatku, náš modul má jednu vadu – používa cudzie typy inak, ako len prostredníctvom rozhraní. Moduly referenčnej politiky to robiť nesmú z dôvodu, aby bolo možné udržať v politike poriadok a prehľad o tom, čo sa s typmi deje. Moduly preto definujú rôzne rozhrania – šablóny, pomocou ktorých umožňujú obmedzenú manipuláciu so svojimi typmi, rolami a inými prvkami politiky. Výhodou je, že pokiaľ sa zachová sémantika rozhraní, je v podstate možné zmeniť vnútornú implementáciu modulu bez toho, aby sa tým narušila kompatibilita alebo funčnosť iných modulov. V prípade modulu výlučne na lokálne použitie si môžeme toto pravidlo

dovoliť porušiť, ale my si teraz ukážeme, ako by náš modul vyzeral korektnejšie.

Problémom bolo použitie roly *staff_r* a domény *staff_t*. Toto použitie preto z nášho modulu odstránime. Namiesto toho definujeme rozhrania, pomocou ktorých môže modul definujúci rolu určiť, že táto rola bude mať povolenú našu doménu, a modul definujúci doménu určiť, že z jeho domény sa má automaticky prechádzať do našej domény spustením programu nášho typu. Doplníme tiež súbor s definíciou bezpečnostného kontextu pre naše tri programy v */usr/local/bin*.

Náš modul sa teda bude skladať už z troch súborov – *myapp.te, myapp.if* a *myapp.fc*:

myapp.te

```
policy_module(myapp, 1.0.7)

type myapp_t;
type myapp_exec_t;
type myapp_files_t;

application_domain(myapp_t, myapp_exec_t)
domain_use_interactive_fds(myapp_t)
userdom_use_inherited_user_terminals(myapp_t)
files_read_etc_files(myapp_t)
selinux_get_fs_mount(myapp_t)
seutil_read_config(myapp_t)
miscfiles_read_localization(myapp_t)
corecmd_exec_bin(myapp_t)
allow myapp_t self:process setpgid;

files_type(myapp_files_t)
allow myapp_t myapp_files_t:dir *;
allow myapp_t myapp_files_t:file *;
```

myapp.if

```
interface(`myapp_role',`

  gen_require(`
    type myapp_t;
  ')

  role $1 types myapp_t;

')

interface(`myapp_autotrans',`

  gen_require(`
    type myapp_t;
    type myapp_exec_t;
  ')

  domtrans_pattern($1, myapp_exec_t, myapp_t)
')
```

myapp.fc

```
/usr/local/bin/hw        gen_context(system_u:object_r:myapp_exec_t,s0)
/usr/local/bin/my.*      gen_context(system_u:object_r:myapp_exec_t,s0)
```

V súbore *myapp.if* sme definovali rozhranie *myapp_role* s jedným parametrom, ktorým musí byť rola, ktorej udelíme právo použiť našu doménu, a rozhranie *myapp_autotrans* tiež s jedným parametrom, ktorým je doména, z ktorej sa bude dať do našej domény prejsť spustením našich programov.

V súbore *myapp.fc* sme definovali, že súbor */usr/local/bin/hw* a všetky súbory, ktoré začínajú reťazcom */usr/local/bin/my* majú mať kontext *system_u:object_r:myapp_exec_t:s0*. Makro *gen_context* vytvorí kontext bez MLS rozsahu, ak politika nepodporuje MLS a kontext s MLS rozsahom, ak politika podporuje MLS. Vďaka tomu nemusíme náš modul upravovať do rôznej podoby v závislosti od toho, či má byť použity v politike s alebo bez podpory MLS.

Modul v tejto podobe je bez problémov skompilovateľný a inštalovateľný do politiky, ale nebude správne fungovať. Čo bude v politike totiž chýbať, sú pravidlá pre povolenie domény pre rolu a pre povolenie prechodu medzi doménami. Ak by náš modul bol súčasťou referenčnej politiky, tak by bolo potrebné upraviť príslušné moduly tak, aby použili riadky typu:

```
myapp_role(staff_r)
myapp_autotrans(staff_t)
```

Keďže my ale nechceme do referenčnej politiky zasahovať, vytvoríme si jeden lokálny modul, ktorý nebude dodržiavať pravidlá o nepoužívaní cudzích rol, typov, a pod., a ktorý zabezpečí povolenie našej domény vybraným rolám a povolenie prechodu z iných domén do našej domény:

myapp_trans.te

```
policy_module(myapp_trans, 1.0.0)

require{
  role staff_r;
  role sysadm_r;
  attribute userdomain;
}

myapp_role(staff_r)
myapp_role(sysadm_r)
myapp_autotrans(userdomain)
```

Po skompilovaní a nainštalovaní aj tohto modulu bude všetko fungovať tak, ako očakávame. Tento modul umožní rolám *staff_r* a *sysadm_r* používať našu doménu a zabezpečí, že pri spustení nášho

programu z akejkoľvek používateľskej domény (domény s atribútom *userdomain*) dôjde automaticky k prechodu do našej domény.

Keďže náš modul obsahuje aj definíciu bezpečnostných kontextov pre súbory, môžeme si ich použitím príkazu *semanage fcontext* skontrolovať:

```
root@debian10:/home/admin/myapp# semanage fcontext -l |grep myapp
/usr/local/bin/hw     all files          system_u:object_r:myapp_exec_t:s0
/usr/local/bin/my.*   all files          system_u:object_r:myapp_exec_t:s0
```

Vidíme, že sme zabudli špecifikovať bezpečnostný kontext pre adresár */mydata*, v ktorom chceme mať naše vlastné údaje chránené pred priamym prístupom používateľa. To môžeme ľahko doplniť aj „zvonka“ tiež použitím príkazu *semanage fcontext*:

```
root@debian10:/home/admin/myapp# semanage fcontext -a -t myapp_files_t -f d /mydata
root@debian10:/home/admin/myapp# semanage fcontext -a -t myapp_files_t '/mydata/.*'
root@debian10:/home/admin/myapp# semanage fcontext -l -C
SELinux fcontext              type               Context

/mydata                       directory          system_u:object_r:myapp_files_t:s0
/mydata/.*                    all files          system_u:object_r:myapp_files_t:s0
```

5.8. Ďalšie zdroje

Ako sme videli v tejto kapitole, subsystém SELinux predstavuje veľmi flexibilný a silný bezpečnostný mechanizmus. Na druhej strane, tvorba SELinux-ových politík nie je práve najjednoduchšia. Vďaka existencii referenčnej politiky je však rozhodne použiteľný v mnohých bežných nasadeniach OS Linux.

Veľmi cenným zdrojom informácií o SELinux-e sú WWW stránky projektu na adrese

```
http://selinuxproject.org/
```

a manuálové stránky jednotlivých programov.

6. Subsystém AppArmor

Subsystém AppArmor je ďalším bezpečnostným modulom povinného riadenia prístupu, ktorý umožňuje obmedzovať oprávnenia jednotlivých programov. V porovnaní s modulom SELinux kontroluje menšie množstvo operácií, čo prispieva aj k menšej náročnosti jeho konfigurácie.

Obmedzenia jednotlivých programov sú definované tzv. *profilom* programu. Ak nejaký program nemá definovaný profil (resp. je neaktívny), jeho operácie nie sú modulom AppArmor obmedzované. Ak program má definovaný aktívny profil, bude môcť vykonávať len operácie, ktoré sú v jeho profile povolené.

Profily jednotlivých programov sú uložené v adresári */etc/apparmor.d* a majú podobu textových súborov. Tieto sú následne pomocou nástroja *apparmor_parser* konvertované do binárnej podoby a nahrávané do jadra operačného systému. Profily sa k jednotlivým programom priraďujú na základe cesty k spustiteľnému programu v súborovom systéme.

Profily umožňujú kontrolovať niekoľko typov operácií (a tento zoznam sa postupne v nových verziách rozširuje):

- súborové operácie,
- použitie *capabilities*,
- použitie sieťovej komunikácie (momentálne nefunkčné).

Typická ukážka jednoduchého profilu pre program môže vyzerať nasledovne:

```
#include <tunables/global>
profile ping /{usr/,}bin/{,iputils-}ping {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,
  network inet raw,
  network inet6 raw,

  /{,usr/}bin/{,iputils-}ping mixr,
  /etc/modules.conf r,

  # Site-specific additions and overrides. See local/README for details.
  #include <local/bin.ping>
}
```

Direktíva *#include* slúži na vloženie častí profilu z iných súborov, ktoré obsahujú rôzne často využívané pravidlá (súbory v podadresári *abstractions*) alebo základné spoločné parametre (napr. súbor (*tunables/global*). Profil začína kľúčovým slovom *profile*, názvom profilu, a cestou k súboru (alebo viacerým súborom), pri spustení ktorého sa má profil aplikovať. Ak je profil priradený len jednému súboru, jeho meno a kľúčové slovo *profile* sa môže vynechať. Niektoré profily predpokladajú možnosť, že administrátor systému bude chcieť nejaké pravidlá do profilu pridať, a preto použitím direktívy *#include* vkladajú aj súbor z podadresára *local*, ktorý slúži na uloženie takýchto vlastných doplnení bez obáv, že pri aktualizácii balíka s profilom dôjde k strate lokálnych úprav. Súbory s profilmi sú často pomenované podľa cesty k programu s tým, že / sa nahradia bodkami, takže napr. vyššie uvedený profil je v súbore */etc/apparmor.d/bin.ping*, no toto pomenovanie nie je pre funkčnosť podstatné.

Cesty k súborom v profiloch, či už pri definovaní programu, ku ktorému je profil priradený, alebo pri definícií ciest k súborom pre povolené operácie, môžu používať niekoľko zástupných znakov:

- * - umožňuje nahradiť ľubovoľný počet znakov iných ako /,
- ** - umožňuje nahradiť ľubovoľný počet ľubovoľných znakov (teda vrátane /),
- ? - umožňuje nahradiť jeden znak iný ako /,
- [] - umožňuje nahradiť jeden zo znakov uvedených medzi [],
- [^] - umožňuje nahradiť jeden znak iný ako uvedené medzi [^ a],
- {} - umožňujú uviesť niekoľko čiarkou oddelených reťazcov predstavujúcich jednotlivé alternatívy v ceste.

Ak meno súboru končí znakom /, ide výlučne o adresár. Text za znakom #, okrem direktívy *#include*, predstavuje komentár.

6.1. Povolenie súborových operácií

Povolenie súborových operácií v profile sa skladá z absolútnej cesty k predmetným súborom alebo adresárom (často s využitím vyššie spomínaných zástupných znakov), popisu oprávnení a končí sa čiarkou (,). Teda napríklad:

```
/etc/** r,
/bin/* r,
```

povoľuje čítanie všetkých súborov v celom podstrome /etc a v adresári /bin (ale nie v jeho prípadných podadresároch).

Povolené základné súborové operácie sú definované nasledujúcimi znakmi:

- r – read – čítanie zo súboru,
- w – write – zápis so súboru, vytvorenie súboru, vymazanie súboru,
- a – append – zápis na koniec súboru (zahrnuté vo w),
- m – mmap – namapovanie súboru do adresného priestoru,
- k – lock – zamknutie súboru (spolu s r alebo w),
- l – link – vytvorenie hardlinku (zároveň sa požaduje, aby vytvorením linku neboli získané práva navyše).

Vytváranie hardlinkov je pri používaní AppArmor-u citlivá operácia, nakoľko oprávnenia sú viazané na cesty k súborom.

V prípade adresárov predstavuje právo r oprávnenie na čítanie obsahu adresára a právo w na zmenu atribútov adresára. Na rozdiel od klasických prístupových práv, AppArmor nepozná právo na použitie adresára v ceste (samozrejme, klasické prístupové práva a ACL sú stále kontrolované štandardne).

Podmienky pre spúšťanie programov sú definované niekoľkými znakmi nasledovne:

- ix – spustiť pod aktuálnym profilom (*inherit*) – na proces sa budú vzťahovať obmedzenia aktuálneho profilu bez ohľadu na to, či spúšťaný program má alebo nemá priradený vlastný profil,
- px – spustiť pod vlastným profilom (*profile*) – na proces sa budú vzťahovať obmedzenia profilu priradeného spúšťanému programu,
- cx – spustiť pod vnoreným profilom (*child*) – na proces sa budú vzťahovať obmedzenia vnoreného profilu v aktuálnom profile, ktorý je priradený spúšťanému programu na základe cesty k nemu,
- ux – spustiť v neobmedzenom režime – program bude spustený bez obmedzenia profilom,
- pix – ako px, ale ak profil neexistuje, tak ako ix,
- pux – ako px, ale ak profil neexistuje, tak ako ux,
- cix – ako cx, ale ak profil neexistuje, tak ako ix,
- cux – ako cx, ale ak profil neexistuje, tak ako cx,

- písmená p, c, u môžu byť aj P, C, U – vtedy sa navyše pri spustení vyčistí prostredie programu.

Špecifikácie podmienok pre spúšťanie typu px a cx (a ich varianty) môžu byť navyše doplnené určením konkrétneho profilu, ktorý sa má použiť namiesto profilu určeného na základe cesty k spúšťanému programu:

```
/bin/xyz px -> profil1,
/bin/pqr cx -> profil2,
```

Cesta k súborom môže byť ešte doplnená prefixom *owner*, ktorý navyše vyžaduje, aby proces pristupujúci k súboru mal FSUID rovné vlastníkovi súboru.

6.2. Povolenie použitia capabilities

Ak má mať program obmedzený profilom možnosť použiť niektoré oprávnenia subsystému *capabilities*, musí mu to profil povoľovať. Pravidlá tohto typu sa skladajú z kľúčového slova *capability*, názvu príslušného oprávnenia (malými písmenami a bez prefixu cap_) a je ukončená čiarkou:

```
capability net_raw,
```

6.3. Povolenie sieťovej komunikácie

Ak má mať program obmedzený profilom možnosť použiť sieťovú komunikáciu, musí mu to profil povoľovať. Základné pravidlá na povolenie sieťovej komunikácie pozostávajú z kľúčového slova *network* a následnej špecifikácie domény (napr. *inet*, *inet6*, ...) a typu služby (napr. *stream*, *dgram*) alebo protokolu (napr. *tcp*, *udp*). Ak niektorá zo špecifikácií (doména, protokol) chýba, znamená to, že sú prípustné všetky. Pravidlo je opäť ukončené čiarkou. Napr.:

```
network,
```

povolí všetku sieťovú komunikáciu. Pravidlo:

```
network inet stream,
```

povolí použitie služieb typu stream používajúci IPv4. Pravidlo:

```
network tcp,
```

povolí použitie protokolu TCP s IPv4 aj IPv6.

Žiaľ, v čase písania tejto kapitoly bola kontroloa sieťovej komunikácie minimálne v systéme Debian 10 nefunkčná a všetka sieťová komunikácia bola automaticky povolená.

6.4. Explicitné zakazovacie pravidlá

Profily môžu obsahuvať aj explicitné zakazovacie pravidlá. Tieto majú prednosť pred povoľovacími pravidlami, teda ak je nejaký prístup zakázaný explicitným zakazovacím pravidlom, bude zakázaný aj v prípade, že existujú povoľovacie pravidlá, ktoré by ho povoľovali. Zakazovacie pravidlo má rovnaký tvar ako príslušné povoľovacie pravidlo, ale začína kľúčovým slovom *deny*, pričom v prípade pravidla, ktorým sa zakazuje spúšťanie programu sa použije iba *x* (teda nie *ix*, *cx*, *px*, ...). Napr. pravidlo:

```
deny /sbin/** x,
```

zakáže spúšťanie akýchkoľvek programov z podstromu /*sbin*, aj keby to bolo inými pravidlami povolené.

6.5. Auditovanie prístupov

AppArmor štandardne generuje auditné záznamy pri zamietnutí prístupu. Neplatí to pre prístupy zamietnuté na základe explicitného zakazovacieho pravidla. AppArmor však umožňuje generovať auditné záznamy aj pri povolení prístupu na základe konkrétneho pravidla. Na to slúži kľúčové slovo *audit*, ktoré môže predchádzať pravidlu. Taktiež je možné skombinovať kľúčové slová *audit* a *deny*, aby sme získali explicitné zakazovacie pravidlo, pri ktorom bude tiež generovaný auditný záznam pri zamietnutí operácie.

6.6. Vnorené profily

Pri popise pravidiel pre povolenie spustenia programu sme spomenuli pojem vnoreného profilu (*child profile*). Vnorený profil je profil definovaný v inom profile. Tento slúži v spojitosti s pravidlom využívajúcim oprávnenie *cx* na vytvorenie špecifického profilu, ktorý bude použitý na obmedzenie určeného programu v prípade, že ten bol spustený programom obmedzeným rodičovským profilom. Uvažujme napr. profil:

```
#include <tunables/global>
/usr/local/bin/mybash {
  #include <abstractions/base>
  /data/rw/** rw,
  /{,usr}/bin/ping cx -> mojping,
  /{,usr}/bin/* ix,

  profile mojping {
    #include <abstractions/base>
    /usr/bin/ping r,
    capability net_raw,
  }
}
```

Tento profil pre program */usr/local/bin/mybash* umožňuje spúšťať programy z /bin a /usr/bin v rámci tohto profilu, no špeciálne pri spustení */usr/bin/ping* (alebo */bin/ping*) použije vnorený profil, ktorý programu *ping* umožní využívať oprávnenie cap_net_raw (ak ho má pridelené prostriedkami subsystému *capabilities*).

6.7. Klobúky

Klobúky (*hats*) sú špeciálnym typom vnoreného profilu, ktorý umožňuje vytvárať programy, ktoré počas svojej práce budú potrebovať v rôznom čase rôzne oprávnenia. Pokiaľ by sme im priradili jeden profil, musel by im povoľovať všetko, čo budú potrebovať počas celého svojho behu. Avšak pokiaľ je program napísaný tak, že vo vhodnej chváli použije špeciálne volanie subsystému AppArmor, môžeme vytvoriť špeciálny vnorený profil – klobúk, na ktorý sa bude môcť program cielene prepnúť. V prípade potreby sa následne môže prepnúť späť na svoj štandardný profil. Klobúky sa vyznačujú špeciálnym pomenovaním, ktoré začína znakom ^:

```
profile /cesta/k/programu {
  ...
  ^klobuk1 {
    #pravidlá pre program s týmto „klobúkom"
    ...
  }
}
```

6.8. Podporné nástroje

Konkrétny profil subsystému AppArmor môže byť v jednom z dvoch módov – *enforce* a *complain*. V *enforce* móde skutočne obmedzuje príslušné procesy, v *complain* móde operácie nezakazuje, ale generuje auditné záznamy podľa toho, ako by boli operácie zakázané. Hlavným významom *complain* módu je jednoduchšie ladenie profilov.

Prepnúť profil pre určitý program do enforce módu je možné použitím príkazu:

```
aa-enforce program
```

Prepnúť profil pre určitý program do complain módu je možné použitím príkazu:

```
aa-complain program
```

Profil je možné aj zakázať (deaktivovať), použitím príkazu:

```
aa-disable program
```

Následne je ho možné opätovne aktivovať prepnutím do enforce alebo complain módu.

Kompilácia profilu a jeho nahratie do jadra operačného systému sa vykonáva použitím príkazu:

```
apparmor_parser -a /etc/apparmor.d/subor_s_profilom
```

Ak chceme už nahratý profil vymeniť za novú verziu, namiesto *-a* použijeme *-r*. Ak chceme profil z jadra odstrániť, tak môžeme použiť *-R*. Všetky profily sa automaticky do jadra nahrávajú pri štarte systému pomocou služby *apparmor*. Nové nahratie všetkých aktívnych profilov môžeme vynútiť aj reštartom tejto služby.

Aktuálny stav subsystému AppArmor môžeme zistiť použitím príkazu:

```
aa-status
```

Tento príkaz vypíše zoznam profilov v jednotlivých módoch aj počet procesov, ktoré sú subsystémom AppArmor aktuálne obmedzované.

Podobne ako pri používaní subsystému SELinux, aj pri používaní subsystému AppArmor môžeme použiť prepínač *-Z* pri príkaze *ps* na vypísanie informácie o použitou profile pre jednotlivé bežiace procesy. V tomto výpise slovo *unconfined* znamená, že proces beží bez obmedzení.

6.9. Ďalšie zdroje

Viac informácií o subsystéme AppArmor je možné nájsť na webe

```
http://wiki.apparmor.net/
```

v manuálových stránkach a na weboch jednotlivých distribúcií Linuxu. Treba si uvedomiť, že AppArmor je neustále vyvíjaný a postupne pribúdajú ďalšie oblasti, ktoré je pomocou neho možné kontrolovať.

Pokročilé bezpečnostné mechanizmy OS Linux

Materiál je výstupom Rozvojového projektu Univerzity Komenského a Ministerstva školstva, vedy, výskumu a športu SR č. 002 UK-2-1/2018 – *„Vzdelávanie pre informačnú spoločnosť“* v oblasti Podpora vysokých škôl pri plnení záväzkov prijatých v rámci Národnej koalície pre digitálne zručnosti a povolania SR.

Autor: Jaroslav Janáček
Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra informatiky

Recenzenti: doc. Ing. Pavel Segeč, PhD.
doc. Ing. Anton Baláž, PhD.

Vydavateľ: Univerzita Komenského v Bratislave
Bratislava 2022
1. vydanie, 123 strán

ISBN 978-80-223-5406-6