MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Česká dokumentace pro SELinux

BAKALÁŘSKÁ PRÁCE

Jan Horák

Brno, jaro 2007

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: Mgr. Jan Kasprzak

Poděkování

Chtěl bych poděkovat zejména panu Martinu Stránskému a Mgr. Janu Kasprzakovi za cenné rady při tvorbě této práce a za všechen čas, který mi takto věnovali.

Shrnutí

Tato práce vznikla jako první ucelená česká příručka pro SELinux, která pomůže zejména administrátorovi nebo i jen zainteresovanému uživateli pochopit danou problematiku a následně využít popsané principy při konfiguraci skutečného systému. V jednotlivých kapitolách se věnujeme problémům, které jsou nutné k úspěšnému pochopení principů SELinuxu.

Klíčová slova

SELinux, Type Enforcement, Role Based Access Control, politika, makra m4, doména, typ, bezpečnostní kontext

Obsah

Kapitola 1

Principy SELinuxu

1.1 Úvod

Přes všechnu snahu vývojářů vyvinout bezchybný a bezpečný systém je jisté, že se jim to prozatím nepodaří. Programování je v porovnání s ostatními vědami v počátcích. Systémy jsou natolik složité, že se v nich vždy najde taková část, která umožní znalému útočníkovi do systému vniknout. Těchto pokusů přibývá každoročně asi o 85%[1], nezbývá tedy nic jiného, než si pomoci nějakými podpůrnými prostředky. Jedním z nich je i *NSA Security-Enhanced Linux*, známý spíše pod zkratkou *SELinux*. SELinux je implementací tzv. *povinného řízení přístupu* (anglicky *Mandatory Access Control*, neboli *MAC*). SELinux vynucuje administrátorem definovanou bezpečnostní politiku nad všemi subjekty a objekty v systému. Umožňuje tak nastavit jemnější práva přístupu k datům a zamezit jejich změnám, ať úmyslným či neúmyslným.

1.2 Historie

Během devadesátých let dnes již minulého století spolupracovaly *NSA* (*National Security Agency*) s *Secure Computing System* na vývoji silné a flexibilní architektury s povinným řízením přístupu. Zaměřily se na teoretické základy a charakteristiku této architektury. Společně s Univerzitou v Utahu poté vytvořily prototyp nazvaný *Flask*. Po další spolupráci se společnostmi *Network Associates* a *MITRE* implementovaly tuto architekturu do operačních systémů Linux. Jejich práce byla uveřejněna v prosinci roku 2000 a byla poskytnuta jako open-source produkt.

Několik dalších distributorů Linuxu poté projevilo zájem o podporu SELinuxu v jejich distribucích. Mezi ně patří Red Hat (v Red Hat Enterprise Linuxu) a SUSE v komerčních distribucích Linuxu. SELinux je již standardní součástí nekomerčních distribucí Debian GNU/Linux, Gentoo Linux a Fedora Core (sponzorované společností Red Hat).

1.3 Porovnání s tradičním unixovým modelem

Operační systémy používají některý z mechanizmů řízení přístupu. Mezi dva nejznámější patří *diskrétní* (*volitelné*) *řízení přístupu* (*DAC*, *Discretionary Access Control* [2]), které je použito v klasickém Linuxu, a *povinné řízení přístupu* (*MAC*, *Mandatory Access Control* [3]), které je předmětem SELinuxu. Oba mechanizmy mohou být v systému implementovány zároveň.

1.3.1 Diskrétní řízení přístupu

Princip diskrétního řízení přístupu je založen na *ACL* (*Access Control List*). Každý soubor nebo program má k sobě asociován jeden ACL, kde jsou definována jeho přístupová práva pro uživatele (vlastníka), skupinu a ostatní. Na základě ACL je tedy povolen či zamítnut přístup k objektu.

DAC neposkytuje žádnou ochranu před použitím poškozeného nebo upraveného softwaru. Každý spuštěný program nebo proces běží pod právy uživatele, který jej spustil. Proces má tedy plnou kontrolu nad všemi daty daného uživatele a má možnost je měnit a zapisovat do nich. Například program na prohlížení pošty nepotřebuje mít plný přístup ke všem datům uživatele. Takovými daty mohou být konfigurační nebo jiné citlivé soubory, což je nežádoucí v případě běžného uživatele a katastrofou u superuživatele, který má neomezený přístup k celému systému. Pokud se útočníkovi podaří převzít kontrolu nad procesem, který běží pod právy superuživatele, naskýtá se mu tak možnost získat kontrolu nad celým systémem. Stejné nebezpečí hrozí i pokud má program nastaven *setuid* nebo *setgid* bit na superuživatele. Především z těchto důvodů DAC neposkytuje dostatečnou bezpečnost.

1.3.2 Povinné řízení přístupu

MAC poskytuje plnou kontrolu nad všemi akcemi programů, které běží v tzv. *sandboxu*, jenž omezuje přístup do jiných částí systému, než je samotný prostor sandboxu. Ve spojitosti se SELinuxem se zavádí speciální *sandboxy* nazývané *domény*, znemožňující i programům pod právy superuživatele tuto doménu opustit. Pokud se útočníkovi podaří převzít kontrolu nad nějakým programem, může provádět pouze takové operace, které měl daný program povoleny, a to i v případě programů pod právy superuživatele. Veškeré programy takovéhoto systému se musí řídit administrátorem definovanou politikou a znemožní tak použití poškozeného nebo upraveného softwaru všem uživatelům i superuživateli.
Ve spojitosti se SELinuxem budeme používat tyto pojmy:

- subjekty – aktivní části systému, tedy procesy
- objekty – pasivní části systému, tedy soubory, adresáře, sokety a další

Poznámka Toto rozdělění bude sloužit jen pro naši lepší představu s čím pracujeme. SELinux mezi subjekty a objekty nedělá rozdíl, dívá se na ně jako na typ.

1.4 Instalace

Existují tři způsoby, jak lze získat SELinux:

- jako integrovanou součást linuxové distribuce, kterou instalujeme na počítač
- stáhnutím, zkompilováním a nainstalováním ze zdrojových kódů poskytovaných na stránkách NSA
- nainstalováním ze zdrojových nebo binárních balíčků z domovských stránek dané linuxové distribuce (pro Debian GNU/Linux .deb balíčky, pro Fedora Core, SUSE Linux nebo Red Hat Enterprise Linux jsou to .rpm balíčky)

V případě, že budeme kompilovat SELinux ze zdrojových kódů, je nutné ověřit, zda naše jádro SELinux podporuje. Pokud ano, vše je v pořádku, jinak je potřeba překompilovat i samotné jádro, a to s podporou *NSA SELinux Support*. My použijeme distribuci *Fedora Core 6*, kde je SELinux součástí instalace, tedy i jádro s jeho podporou. Po instalaci stačí pouze aktualizovat jednotlivé části SELinuxu, a to například příkazem v shellu (`yum` případně vyřeší závislosti a aktualizuje i ostatní časti SELinuxu):

```
[root@noutec ~]# yum update libselinux selinux-policy-targeted
```

Budeme ještě potřebovat balíček *selinux-policy-devel*, kde jsou soubory nutné pro kompilaci vlastních pravidel, balíček *selinux-policy-strict* pro striktní politiku, balíček *policycoreutils-newrole* pro změnu rolí a případně *selinux-policy-mls* pro MLS politiku.

```
[root@noutec ~]# yum install selinux-policy-devel \
selinux-policy-strict policycoreutils-newrole selinux-policy-mls
```

1.5 Politika a jádro systému

Politika je množina pravidel, které jsou základem SELinuxu definující všem objektům v systému typ a procesům jejich domény, identitám určuje v jakých rolích mohou operovat a rolím stanovuje, do jakých domén mohou tyto role vstoupit a ke kterým typům mají přístup. Politika je administrátorsky definovatelná, což umožňuje pravidla nakonfigurovat přesně podle potřeb daného systému. Zdrojové kódy pravidel jsou zkompilovány do binární podoby a jsou poté zavedeny do jádra.

V dřívějších verzích distribucí Fedora Core byla celá politika zkompilována do jednoho modulu a jakákoliv změna politiky vedla ke znovukompilaci celého modulu. Ve verzi *Fedora Core 5* se zavádí mechanismus zvaný *LSM* (*Linux Security Modules*), usnadňující práci při konfiguraci. Množiny pravidel jsou zkompilovány do modulů a tyto moduly se mohou za běhu systému libovolně nahrávat či odstraňovat z jádra, aniž by bylo nutné celou politiku opět kompilovat.

1.6 Start systému

Když se bootuje jádro s podporou SELinuxu, je v něm pevně dáno, že má jádro běžet v doméně *kernel_t*. V této době sice jádro má svůj kontext, ale politika ještě nahrána není. Všechny procesy, které jádro spustí, zůstavají v doméně *kernel_t*. Až spuštěný proces *init* načte a nahraje do jádra politiku z */etc/selinux/(strict|targeted|mls)/policy/policy.verze*. Následně se *init* běžící nyní v doméně *init_exec_t* restartuje a podle pravidel v jádře přejde do domény *init_t*, ve které už zůstane.

1.7 Řízení přístupu

Bezpečnost SELinuxu je založena na dvou základních mechanizmech. Primárním je *Type Enforcement* (*TE*), sekundárním je *Role Based Access Control* (*RBAC*) a volitelným *Multi-Level Security modelem* (*MLS*). Pokud bude program provádět nějakou operaci, musí být povolena

těmito mechanizmy. Samozřejmě musí vyhovět i omezením, která jsou kladena diskrétním řízením přístupu, jako jsou běžná přístupová práva. SELinux omezuje jen operace, které při svém konání komunikují s jádrem. Co nejde přes jádro, SELinux neomezuje.

1.7.1 Bezpečnostní kontext (Security Context)

Kontext se většinou skládá z těchto částí:
system_u:system_r:sshd_t

Každý *subjekt* (proces) a *objekt* (například soubor, adresář, soket, paket) má přiřazen *bezpečnostní kontext*. Kontext se skládá ze tří nebo čtyř částí oddělených ':', a to *identity, role, domény* nebo *typu* a případným *mls*.

Kontext pro soubory lze vypsat příkazem `ls --context` nebo `ls -Z`, pro uživatele příkazem `id -Z` a pro proces příkazem `ps -Z`. Kontext se vypisuje ve tvaru *identita:role:doména(typ):mls.*

- **system_u:**system_r:sshd_t

  identita (neboli SELinux uživatel) – se líší od identity v klasickém Linuxu, ačkoliv mohou mít stejnou textovou podobu. Je třeba cítit jejich rozdíl. Identita SELinuxu je součástí bezpečnostního kontextu. Jedna SELinux identita může být přiřazena více uživatelům, kteří mají podobná oprávnění v systému. Na základě identity se rozhoduje, do kterých rolí lze přejít. V targeted politice je tato položka (stejně jako role) v podstatě ignorována. V rámci SELinuxu příkaz `su` identitu nemění:

```
[honza@noutec ~]$ id -Z                    # běžný uživatel
user_u:system_r:unconfined_t
[honza@noutec ~]$ su                       # změna uživatele
[root@noutec ~]# id -Z                     # uživatel root
user_u:system_r:unconfined_t
```

- system_u:**system_r:**sshd_t

  role – na základě role lze blíže specifikovat přístupy k typům

- system_u:system_r:**sshd_t**

  typ – jedná se o nejdůležitější část bezpečnostního kontextu. Na základě typů, na které se vztahují *TE pravidla*, se rozhodují téměř všechny přístupy.

- system_u:system_r:sshd_t:**s0-s15:**c0.c255

  mls – udává citlivost dat, pouze subjekty se stejnou citlivostí mohou k takovým objektům přistupovat (rozsah citlivosti je v rozmezí s0 až s15)

- `system_u:system_r:sshd_t:s0-s15:`**`c0.c255`**

  mcs – udává kategorii (třídu) objektu (rozsah kategorie je v rozmezí c0 až c255)

Například `sshd` daemon má kontext:

```
system_u:system_r:sshd_t:s0-s15:c0.c255
```

System_u je implicitní identita pro systémové procesy a zároveň udává roli *system_r*, která je rovněž implicitní. *Sshd_t* je doména procesu. Pokud otevřeme `ssh` spojení, ukládají se dočasné soubory do */tmp/*, to ale znamená, že musí existovat TE pravidla (přístupová pravidla), která povolí doméně (resp. typu) *sshd_t* číst adresář */tmp/*, vytvářet v něm nové soubory, adresáře, zapisovat do nich, a další operace. MLS část *s0-s15*, pro proces zvaná oprávnění, povoluje přistupovat k objektům, které mají takovouto citlivost. MCS část *c0.c255* jsou kategorie, ke kterým může sshd daemon přistupovat.

Kontext souboru vytvořeného v */tmp/* může být následující:

```
user_u:object_r:user_tmp_t:s0:c0
```

User_u je identita uživatele, *object_r* je role, náležící souborům a *user_tmp_t* je typ pro soubor v */tmp/*, který proces vytvořil. V souvislosti s TE pravidly to znamená, že proces opět smí pracovat s */tmp/* stejně jako sshd výše, ale navíc je tu pravidlo (přechodové pravidlo), že pokud proces vytvoří soubor v */tmp/* (typ *tmp_t*), tento soubor bude typu *user_tmp_t*. MLS část povoluje subjektům s oprávněním *s0* přistupovat k těmto datům. MCS část udává kategorii souboru. Pouze subjekty, které mají ve své MCS části kategorii *c0*, mohou k souboru přistupovat.

1.7.2 Type Enforcement, TE

V této kapitole budeme pracovat s touto částí kontextu:
`system_u:system_r:`**`sshd_t`**

Type Enforcement je srdcem celého SELinuxu. Ať zvolíme kteroukoliv politiku, TE je jejím základem. Každému subjektu (běžícímu procesu) i objektu (např. souboru) je přiřazen typ. U subjektu (procesu) se typ také nazývá doména. Se všemi subjekty (resp. objekty), které jsou ve stejné doméně (resp. mají stejný typ), je zacházeno stejným způsobem. SELinuxu nezáleží na tom, zda se jedná o subjekt nebo objekt, pracuje pouze s jejich typem. TE funguje na dvou základních typech pravidel, na *přechodových (transition)* a *přístupových (access)*.

Hlavní myšlenkou je rozdělit celý systém na menší části, kde by bez použití TE měl v podstatě kdokoliv (uživatel či proces) šanci přistupovat ke všem souborům a procesům, což v žádném případě není žádoucí. Každá z těchto částí bude oprávněna provádět jen specifické, nakonfigurované, operace. Subjektům tedy povolíme jen přesně to, co od nich potřebujeme, nic víc. Doménu autorizujeme pro přístup k některým typům souborů a definujeme, co s nimi může provádět. Pokud si ale představíme složitější subjekt jako je

třeba proces *init*, který určitě potřebuje mít přístup k velkému množství typů, vzniká tu zase velké nebezpečí při jeho diskreditaci a tím přístup k celému systému. Proto raději vytvoříme menší domény pro potřebné operace a v případě potřeby povolíme doméně přechod do jiné domény, která bude mít ještě menší, větší a nebo úplně jiná oprávnění. Každý proces smí v daný okamžik běžet pouze v jediné doméně.

Ačkoliv můžeme použít politiky poskytované v linuxových distribucích, je pravděpodobné, že nebude vyhovovat všem požadavkům kladeným na zabezpečení konkrétního systému. Ze začátku čekají administrátora opravdu perné chvíle. Pokud se mu však podaří vše vhodně nakonfigurovat a napsat dobrá pravidla, získá tak mnohem lépe zabezpečený systém, v němž bude daleko těžší napáchat škody, a kdyby k nabourání do systému přece jenom došlo, bude poškozena jen malá část.

Deklarace atributů

Pokud budeme upravovat již definovaný typ nebo vytvářet nový, musíme explicitně stanovit, co tento typ znamená a k čemu slouží. Tohoto dosáhneme pomocí *atributů*. Atribut sjednocuje množinu typů, které mají podobné vlastnosti. Každý typ může mít několik atributů a atribut může být asociován několika typům. Přiřazením atributu typu řekneme, že tento typ bude představovat například doménu pro proces, spustitelný nebo běžný soubor a mnoho dalšího. V SELinux pravidlech se dá typ a atribut zaměňovat – tam kde můžeme použít typ, můžeme použít i atribut. Přehled některých atributů je v příloze. Dříve než typu atribut přiřadíme, musíme ho nejprve deklarovat, a to následovně:

```
# atribut domain mají všechny typy, které jsou domény \
  pro proces
attribute domain;

# atribut file_type je přiřazen všem typům, které se \
  používají pro soubory
attribute file_type;
```

Deklarace typů

Než začneme psát naše pravidla pro některou část systému, musíme deklarovat typ, se kterým budeme následně pracovat. Nový typ se deklaruje pomocí direktivy *type*, za ní jméno nového typu a následně množina atributů tohoto typu oddělené čárkou.

```
# typ etc_t je typ, který mají soubory a adresáře v /etc/
# atribut sysadmfile značí, že tento typ bude přístupný \
  administrátorovi
# atribut file_type značí, že se jedná o soubor na disku

# type jméno_typu, atributy ;
type etc_t, sysadmfile, file_type;
```

Teď víme, jak vytvořit nový typ a přiřadit mu nějaké atributy, už jen zbývá s ním začít pracovat. Vysvětlíme si dva druhy pravidel: přechodová a přístupová.

Přechodová pravidla pro procesy

Část kontextu, které se zde věnujeme:
`system_u:system_r:`**`sshd_t`**

Přechodová pravidla definují novou doménu pro proces nebo nový typ pro objekt. Nutnost přechodu do nové domény bude nejlepší popsat na následujícím příkladě.

Mějme uživatele s typem *user_t*, který si chce změnit heslo. Soubor s hesly má typ *shadow_t*. Doména oprávněná měnit *shadow_t* je *passwd_t*. Kdybychom povolili uživateli se svým typem přímo měnit soubor s hesly, ničeho bychom nedosáhli. Stejný výsledek by byl, kdybychom typu *user_t* povolili přímý přechod do *passwd_t* domény. Potřebujeme najít nějaký způsob, jak se do domény *passwd_t* bezpečně dostat. Toho dosáhneme právě *přechodovými* pravidly.

Musíme najít vhodný vstupní bod (*entrypoint*) do požadované domény. Tímto bodem bývá spustitelný soubor. V našem příkladě to bude příkaz `/sbin/passwd` s typem *passwd_exec_t* pro změnu hesel. `/sbin/passwd` autorizujeme ke vstupu do domény *passwd_t*.

Postupně tedy: uživateli s typem *user_t* povolíme spustit `/sbin/passwd` s typem *passwd_exec_t*. Jelikož tento typ slouží jako *vstupní bod* do domény, přejde se do domény *passwd_t* a uživatel si smí změnit heslo.

Až budeme konfigurovat systém a psát přechodová pravidla, musíme provést následující kroky:

- vytvoříme typ pro spustitelný soubor pro vstup do domény (atribut *exec_type*) a doménu (atribut *domain*)

```
# typ passwd_t je doména pro proces
type passwd_t, domain;

# passwd_exec_t je typ spustitelného (exec_type) souboru \
  na disku (file_type)
type passwd_exec_t, exec_type, file_type;
```

- povolíme subjektům s určitým typem (např. *user_t* pro uživatele) spustit soubor

```
# subjekt s typem user_t může spustit (execute) soubor \
  s typem passwd_exec_t
# aby soubor mohl spustit, musí mít povoleno číst (read) \
  tento soubor, a aby soubor mohl číst, musí mít povoleno \
  zjistit atributy (getattr) souboru
allow user_t passwd_exec_t: file { read getattr execute };
```

- povolíme spuštěnému programu přejít do požadované domény

```
# type_transition definuje přechod
# po spuštění souboru s typem passwd_exec_t přejde \
  doména s typem user_t do nové domény passwd_t
type_transition user_t passwd_exec_t: process passwd_t;

# pravidlo povoluje provést přechod procesu mezi doménami
# doména user_t může přejít do domény passwd_t
allow user_t passwd_t: process transition;

# typ passwd_t může spustit (execute) soubor s typem \
  passwd_exec_t
# ke spuštění je potřeba číst (read) soubor a zjistit \
  atributy (getattr) souboru
# operace entrypoint umožní po spuštění souboru \
  s typem passwd_exec_t přechod do nové domény passwd_t
allow passwd_t passwd_exec_t: file { read entrypoint \
                                     execute getattr };
```

Přechodová pravidla pro soubory

Část kontextu, které se věnujeme:
`user_u:object_r:`**`sshd_tmp_t`**

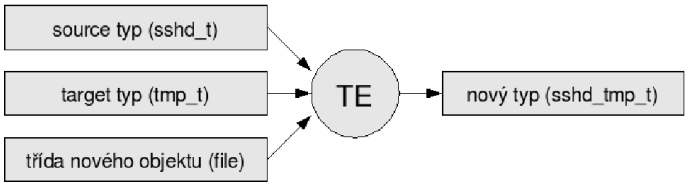
Druhým typem přechodových pravidel nastavíme výsledné typy souborů, pokud je vytvoří nějaká doména. Tato pravidla si můžeme představit jako dědění. Když vytvoříme v adresáři nový soubor, zdědí svůj typ od nadřazeného adresáře.

Přechodová pravidla se skládají z nového typu, jeho třídy a dvou vstupních typů. Zdrojový typ (*source*), což je typ domény, a cílový typ (*target*), což je typ, ke kterému doména přistupuje.

Pravidlo

```
type_transition sshd_t tmp_t: file sshd_tmp_t;
```

říká, že pokud proces v doméně *sshd_t* (source) vytvoří soubor (třída file) v adresáři *tmp_t* (target), tak typ tohoto souboru má být *sshd_tmp_t* (nový typ). Schéma tohoto pravidla je na obrázku.



Obrázek 1.1: Přechodová pravidla

Přístupová pravidla

Přístupy jsou prováděny na základě této části kontextu:
`system_u:system_r:sshd_t`

Dostáváme se k nejdůležitější části, kterou jsou *přístupová pravidla*. Těmito pravidly přesně definujeme *kdo*, resp. *jaký typ nebo atribut (typy s tímto atributem)*, smí přistupovat *k jinému typu*, který představuje nějakou *třídu* (soubor, adresář a další), a které *operace* má povoleno provádět. Jestliže se subjekt bude snažit provádět operace s nějakým typem a třídou, zkontroluje se, jestli k dané operaci existuje pravidlo. Neexistuje–li, operace je zamítnuta a auditována. TE se drží přístupu: co není explicitně povoleno, je zamítnuto.

Na vstup přichází oprávnění pro subjekt, typ subjektu, typ objektu, třída objektu a operace.
Možná oprávnění pro subjekt jsou:

- allow – povoluje subjektu provádět s objektem definované operace

  ```
  allow sshd_t sshd_exec_t: file { read execute getattr };
  ```

  Povoluje procesu v doméně *sshd_t* číst a spouštět soubor s typem *sshd_exec_t*. Pokud se subjekt snaží provádět s objektem operaci, jež není povolena, je tato operace logována. Oprávněná operace logována není.

- auditallow – povoluje pouze logování, nepovoluje samotnou operaci (to může pouze *allow*)

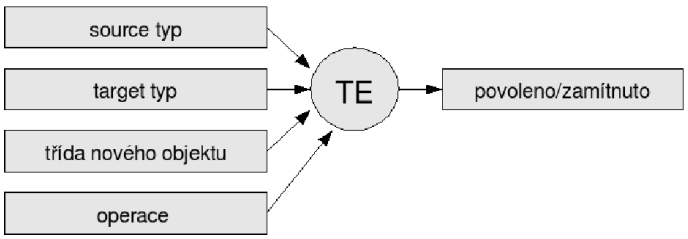
  ```
  auditallow unconfined_t security_t: security \
  { load_policy setenforce };
  ```

  Jestliže je procesu v doméně *unconfined_t* povoleno nahrávat politku SELinuxu do jádra a nahraje ji, v logu se objeví zpráva *avc: granted { load_policy }.*

- dontaudit – neoprávněné operace se nelogují, zamezí se tak zbytečnému plnění logu

  ```
  dontaudit named_t root_t: file { read };
  ```

  Neoprávněný pokus domény *named_t* číst soubor s typem *root_t* není auditovám.



Obrázek 1.2: Schéma přístupových pravidel

Opakem přístupových pravidel jsou tzv. *assertion pravidla*. Těmito pravidly explicitně definujeme akce, které nesmí být povoleny. Tato pravidla lze použít při hledání chyb, které vznikají například při použití *maker*. Tato makra mohou povolovat i operace, které chceme zakázat. Pokud takto některou operaci zakážeme a existují pravidla povolující tuto operaci, assertion pravidlo je překryje, tedy zakáže. Pravidla vypadají stejně jako přístupová, pouze začínají direktivou *neverallow*. Následující pravidlo znemožní doménám (typy s atributem *domain*) pokus o přechod do typu, který není doména.

```
# znak '~' reprezentuje doplněk dané množiny
# zamezí doménám (typy s atributem domain), aby se pokusily \
  změnit svůj typ, na typ, který nepředstavuje doménu (typ \
  nemá atribut domain)
neverallow domain ~domain: process transition;
```

1.7.3 Role Based Access Control, RBAC

V této kapitole se věnujeme následující části kontextu:
`system_u:`**`system_r:`**`sshd_t`

RBAC je přístup založený na rolích. Pomocí RBAC docílíme ještě dalšího rozdělení oprávnění v systému. Vytvoříme novou roli nebo upravíme dříve definovanou a povolíme, k jakým typům (doménám) smí přistupovat a jaké operace s nimi může provádět. Tento způsob přístupu se v praxi příliš nepoužívá, proto se mu zde nebudeme dále věnovat. Uživatel, pokud je mu povoleno, může vystupovat v několika rolích, ale stejně jako u TE domén, smí v jeden okamžik vystupovat pouze v jedné roli.

Ukážeme si, jak vypadají RBAC pravidla:
Pravidly tohoto typu

```
# user identita roles { množina rolí }
user staff_u roles { staff_r };
```

povolíme identitám *staff_u* vystupovat v roli *staff_r*.

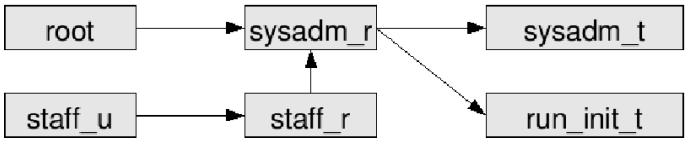
Následujícím typem pravidel

```
# role naše_role typem { množina_typů }
role sysadm_r types { sysadm_t run_init_t };
```

povolujeme přístupy role *sysadm_r* k typům *sysadm_t* a *run_init_t*.

A konečně, takto povolujeme přechod jedné role do jiné, z role *staff_r* do role *sysadm_r*.

```
# allow naše_role { množina_rolí }
allow staff_r { sysadm_r };
```



Obrázek 1.3: Změna role

1.7.4 Constraints (omezení)

Constrains pravidla se mohou rozhodovat podle všech částí kontextů:

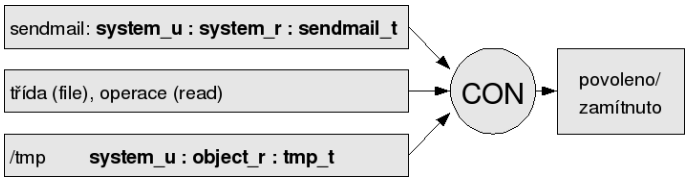
```
honza:user_r:user_t
   ↕      ↕      ↕
root:user_r:user_t
```

Constraints jsou dodatečným *zpřísněním* TE a RBAC přístupových pravidel, implementované pomocí *neverallow* pravidel. V podstatě se jedná o jednoduché *if-then-else* konstrukce, kde se na základě dvojice (brány jako source a target stejně jako v TE) *identit*, *rolí* nebo *typů* (převážně jejich *atributů*) rozhoduje, má–li být oprávnění provádět danou *akci* skutečně uděleno. Identity jsou reprezentovány jako *u1* a *u2*, role jako *r1* a *r2* a typy jako *t1* a *t2*.

`u1:r1:t1` a `u2:r2:t2` jsou kontexty, které se pomocí contstraints kontrolují.

Poznámka Kvůli jednoznačnosti byl ponechán původní název *constraints* místo českého překladu *omezení*.

Než se povolí doméně s typem *sendmail_t* číst (read) soubor (file) s typem *tmp_t*, zkontrolují



Obrázek 1.4: Mechanizmus constraints pravidel

se *constraints pravidla* a ty povolí nebo zakáží operaci. Tato omezení mají v SELinuxu zásadní roli. Těmito pravidly zamezíme, aby se například doména snažila přejít do typu, který nepředstavuje doménu, přepnutí do libovolné identity, role nebo typu a nebo aby vytvořený soubor měl typ, který je pro proces.

Zapisují se tvaru (třída i operace může být zadána makrem):

```
# constrain { množina_tříd } { množina_operací } ( podmínky );
constrain process transition
   ( u1 == u2
   or (t1 == privuser and t2 == userdomain )
   or (t1 == crond_t and t2 == user_crond_domain)
   or (t1 == userhelper_t)
   or (t1 == priv_system_role and u2 == system_u ) );
```

Procesu je povoleno změnit doménu, pokud je splněna alespoň jedna z podmínek:

- zůstane zachována identita
- source typ má atribut *privuser* (atribut opravňující změnit identitu) a target typ má atribut *userdomain* (atribut pro uživatelské domény, např. user_t nebo staff_t)
- source typ je *crond_t* a target typ má atribut *user_crond_domain* (tento atribut mají pouze typy user_crond_t a sysadm_crond_t)
- source typ je *userhelper_t*
- source typ má atribut *priv_system_role* (atribut umožňující změnit identitu na *system_u* a roli na *system_r*) a target identita je *system_u*

Pokud není splněna žádná z těchto podmínek, procesu není povoleno změnit doménu.

1.7.5 MLS

Zde napsaná pravidla pracují s touto částí kontextu:
`system_u:system_r:sshd_t:`**`s0-s15`**`:c0.c255`

Během vývoje Fedora Core 5 a Red Hat Enterprise Linuxu byla vyvinuta nová bezpečnostní politika, hlavně pro použití na serverech, kombinující *Type Enforcement* mechanizmus a *Bell-LaPadula* model (principy *no-read-up* a *no-write-down*). Politika zachovává předchozí principy, ale přidává další komponentu bezpečnostního kontextu, a to *bezpečnostní úroveň* (*security level*) – skládající se z *citlivosti* (*sensitivity*) a *kategorie* (*category*).

Citlivost nabývá hodnot mezi *s0–s15* a je hierarchicky uspořádaná s tím, že *s0* (např. nezařazená data) představuje nejnižší a *s15* (přísně tajná data) nejvyšší citlivost. Kategorie je v rozmezí *c0–c1023* (např. chemie, fyzika, lékařství). Jedné bezpečnostní úrovni může být přiřazena jedna citlivost a žádná nebo více kategorií. Bezpečnostní úroveň přiřazená objektu je brána jako *klasifikace* a subjektu jako *oprávnění*. Hlavní myšlenkou MLS tedy je povolit či zamítnout subjektu běžícímu s nějakým oprávněním přístup k objektu s určitou klasifikací.

Rozhodování o přístupech na základě MLS je prováděno pomocí *constraints pravidel*. Jsou sice zachovány dvojice identit, rolí i typů, ale rozhoduje se převážně podle přibývající dvojice *l1*, *l2* (*jako low*) a *h1*, *h2* (*jako high*). Pokud si vezmeme například MLS část *s0-s3:c0.c15*, tak *s0:c0.c15* je *low* část a *s3:c0.c15* je *high* část oprávnění (pomlčka a tečka značí rozsah).

Constraints pravidla (v případě MLS psáno jako *mlsconstrain*)

```
mlsconstrain { dir file lnk_file chr_file blk_file sock_file \
               fifo_file } { read getattr execute }
     (( l1 dom l2 ) or
      (( t1 == mlsfilereadtoclr ) and ( h1 dom l2 )) or
      ( t1 == mlsfileread ) or
      ( t2 == mlstrustedobject ));
```

```
mlsconstrain { file lnk_file fifo_file dir chr_file blk_file \
               sock_file } { write create setattr relabelfrom \
               append unlink link rename mounton }
     (( l1 eq l2 ) or
      (( t1 == mlsfilewritetoclr ) and ( h1 dom l2 ) \
         and ( l1 domby l2 )) or
      ( t1 == mlsfilewrite ) or
      ( t2 == mlstrustedobject ));
```

První pravidlo povolí číst a spustit soubory, pokud je splněna alespoň jedna z podmínek:

- oprávnění subjektu je nadmnožinou objektu (to znamená: *l1* > *l2*, například vyhovuje **s1**-s2:c1 a **s0**-s2:c1)
- typ subjektu má atribut *mlsfilereadtoclr* a *h1* je nadmnožinou *l2* (s2-**s15**:c5 a **s5**:c5)
- typ subjektu má atribut *mlsfileread* (subjekt s tímto atributem smí číst i soubory s vyšší citlivostí)
- typ objektu má atribut *mlstrustedobject* (do objektů s tímto atributem smí zapisovat všechny subjekty bez ohledu na své oprávnění)

Druhé pravidlo povolí subjektu zapisovat, vytvářet, nastavovat atributy, přeznačkovat a další operace na souborech nebo adresářích, pokud je splněna alespoň jedna z podmínek:

- *low hranice* oprávnění subjektu je stejná jako *low hranice* citlivosti objektu (s0 a s0, s1 a s1 atd.)
- typ subjektu má atribut *mlsfilewritetoclr* a *high hranice* subjektu je nadmnožinou *low hranice* objektu a *low hranice* subjektu je podmnožinou *low hranice* objektu (kupříkladu **s2**-**s5**:c0.c6 a **s3**:c3)
- typ subjektu má atribut *mlsfilewrite* (subjekt s tímto typem smí zapisovat do souborů s nižší citlivostí)
- typ objektu má atribut *mlstrustedobject*

Pro konfiguraci MLS částí použijeme nástroj `semanage`, pomocí kterého nadefinujeme rozsahy pro jednotlivé uživatele nebo SELinux identity a přeznačkujeme jejich domovské adresáře. Například

```
[root@noutec ~]# semanage user -m -r s0-s3:c0.c25 staff_u
[root@noutec ~]# chcon -l s0-s3:c0.c25 /home/
```

Změníme oprávnění pro SELinux identity *staff_u* na *s0-s3:c0.c25*. K datům, které nespadnou do tohoto rozsahu, nebudou tito uživatelé oprávněni přistupovat. Posléze mohou *staff_u* uživatelé libovolně přeznačkovat své soubory a rozdělit tak svá data.

1.7.6 MCS

Odstavec zaměřený na tuto část kontextu:
`system_u:system_r:sshd_t:s0-s15:c0.c255`

MCS politika vznikla úpravou MLS politiky využívající základní kostru MLS, včetně označkování a MLS kódu v jádře, ale odstraňuje některé části z MLS. MCS v podstatě ignoruje citlivost dat, vše je označkováno s citlivostí *s0*. Nepoužívá Bell-LaPadula model, neřeší se tedy pronikání ukládaných dat s vyšší citlivostí do nižší citlivosti, viditelné třeba pokud administrátor ukládal data do */tmp/*.

Uživatel svým datům přiřazuje *kategorii* a pouze subjekty, které mají stejnou kategorii, mohou k těmto datům přistupovat. MCS politika se podobá diskrétnímu přístupu, při rozhodování o přístupu či zamítnutí přístupu k datům se nejprve kontroluje DAC a TE přístup a až poté MCS. Implicitně jsou uživatelé oprávněni přistupovat ke všem kategoriím, což administrátor může samozřejmě změnit.
Konfigurace rozsahů MCS politiky se provádí stejným způsobem jako v MLS politice, s jediným rozdílem, že je použita citlivost *s0*.

```
 mlsconstrain file { read }
(( h1 dom h2 ) or ( t2 == domain ) or ( t1 == mlsfileread ));

 mlsconstrain file { create relabelto }
(( h1 dom h2 ) and ( l2 eq h2 ));
```

První pravidlo povoluje subjektu číst soubor, jestliže je splněna jedna z podmínek:

- oprávnění subjektu (h1) je nadmnožinou klasifikace souboru (pokud má h1 oprávnění c0, **c6**, c9 a klasifikace souboru je například **c6**, soubor může přečíst)
- target typ má atribut *domain*
- typ subjektu má atribut *mlsfileread*

Druhé pravidlo povoluje subjektu vytvořit nebo přeznačkovat soubor, pokud jsou splněny obě tyto podmínky zároveň:

- oprávnění subjektu je nadmnožinou klasifikace souboru (s0:**c0.c15** a s0:**c7**)
- citlivost dat je zachována (tzn. **s0** a **s0**)

Kapitola 2

SELinux a Fedora Core 6

V této kapitole si ukážeme, kde jsou uloženy jednotlivé soubory SELinuxu, co obsahují, k čemu slouží a jaké je nastavení SELinuxu ve Fedora Core 6. Všechny důležité soubory se nachází v adresářích */etc/selinux/*, */usr/share/selinux/* a */selinux/*.

2.1 /etc/selinux/

V */etc/selinux/* jsou adresáře pojmenované podle politik nainstalovaných v systému, kde jsou uloženy soubory pro jednotlivé politiky, a konfigurační soubor SELinuxu. V souboru */etc/selinux/config* je globální nastavení SELinuxu, můžeme zde nastavit v jakém *módu* a jaká *politika* bude v systému po startu aktivní. Podle výpisu následujícího konfiguračního souboru je v nastavena *targeted* politika (*SELINUXTYPE*) a *enforcing* mód SELinuxu (*SELINUX*), soubor je možno ručně editovat.

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#    enforcing - SELinux security policy is enforced.
#    permissive - SELinux prints warnings instead of enforcing.
#    disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= type of policy in use.
#              Possible values are:
#    targeted - Only targeted network daemons are protected.
#    strict - Full SELinux protection.
SELINUXTYPE=targeted
# SETLOCALDEFS= Check local definition changes
SETLOCALDEFS=0
```

Módy SELinuxu:

- enforcing – v tomto módu se vynucují nakonfigurovaná pravidla SELinuxu. Všechny operace jsou zakázány, pokud nebyly explicitně povoleny. Každý pokus o nějakou operaci se loguje do */var/log/audit/audit.log* nebo */var/log/messages* jako *AVC zprávy*. Tento mód by měl být nastaven implicitně pro jakýkoliv běžící systém.
- permissive – mód, ve kterém se logují AVC zprávy, ale všechny operace jsou povoleny. Oproti enforcing módu jsou v permissive módu dva základní rozdíly: logují se pouze první pokusy o přístupy dané domény k nějakému objektu a v permissive módu se

mohou objevit AVC zprávy, které by se v enforcing módu nelogovaly. Například určité doméně není povoleno číst adresář a soubory v něm, v enforcing módu se generuje AVC, že není povolen přístup k adresáři, zatímco v permissive módu se generuje AVC i pro pokusy o přístupy k jednotlivým typům souborů v tomto adresáři. Permissive mód by měl být aktivní pouze v případě, že je potřeba doladit, upravit nebo přidat nějaká pravidla, protože ačkoliv SELinux běží, pravidla jsou ignorována.

- disabled – SELinux je vypnut a všechny nové objekty jsou vytvořeny bez bezpečnostního kontextu. Pro vypnutí SELinuxu je potřeba editovat */etc/selinux/config* a restartovat systém, protože SELinux je stále v jádře. Pro znovuzapnutí SELinuxu je nutné přeznačkovat celý souborový systém, aby se správně vytvořily kontexty.

Politiky SELinuxu:

- strict – striktní politika byla prvotní politikou, která byla NSA poskytnuta a byla implicitně používána ve Fedora Core. Tato politika v podstatě uzamyká celý systém. Omezení jsou kladena na celý systémový i uživatelský prostor. Během vývoje se však ukázalo, že je tato politika vhodná spíše pro systém, kde mohou uživatelé například jen přistupovat k internetu a ukládat soubory do určené složky. S ohledem na množství aplikací by vznikalo velké množství pravidel, které by bylo třeba nakonfigurovat. Nástin takové konfigurace si ukážeme později při konfiguraci prohlížeče Firefox.

- targeted – po předchozích zkušenostech byla snaha najít vhodný kompromis a tím byla právě tato politika, zabezpečující pouze služby, které jsou připojeny k síti (například apache), jelikož tyto části systému jsou potenciálními místy útoku hackerů. V targeted politice nemá identita a role význam. Uživatelské akce jsou prováděny v doméně *unconfined_t*, na kterou nejsou kladena žádná omezení, takové procesy mají stejný přístup k systému, jako kdyby byl SELinux vypnut.

- mls – politika kladoucí další omezení na základě citlivosti a kategorie dat (viz. MLS), politika bývá implementována jako kombinace se striktní nebo targeted politikou

2.1.1 /etc/selinux/(targeted|strict|mls)

seusers
V seusers je nastaveno mapovaní uživatelů do SELinux identit, úroveň oprávnění a kategorie pro indentitu. Pokud pro přihlašujícího se uživatele není explicitně dáno, jaká SELinux identita mu má být přiřazena, je použita *default* identita.

Zápis je tvaru *uživatel:indentita:oprávnění:kategorie*. Soubor je editovatelný pomocí programu `semanage`.

```
system_u:system_u:s0-s0:c0.c1023
root:root:s0-s0:c0.c1023
__default__:user_u:s0
```

setrans.conf
Nejedná se přímo o konfigurační soubor, který by nějak ovlivňoval chování SELinuxu, ale

můžeme zde nadefinovat překlady *MLS:MCS* do čitelnější podoby. Pokud editujeme soubor a zapíšeme:

```
s0:c0=TopSecretData
```

Při vypsání kontextu souboru se nevypíše jako poslední komponenta *s0:c0*, ale přeloží se jako *TopSecretData*.

2.1.2 /etc/selinux/(targeted|strict|mls)/contexts/

default_type
Zde jsou nastaveny implicitní typy pro jednotlivé role. Pokud ve strict nebo MLS politice použijeme příkaz `newrole -r sysadm_r`, bude uživateli přiřazen typ *sysadm_t*.

default_contexts
Specifikuje implicitní role pro přihlášení uživatelů, ssh sezení nebo cron práce (job). V prvním sloupci je část kontextu (role:doména) procesu. Když je třeba přihlášení, sezení nebo jobu přiřadit kontext, najde se v prvním sloupci vyhovující část a přiřadí uživateli první vyhovující část kontextu z druhého sloupce, pro který je uživatel autorizován. V prvním sloupci nenajdeme záznam *user_r:user_t*, protože tito uživatelé nejsou oprávněni svůj kontext změnit.

failsafe_contexts
Pokud se systému nepodaří najít kontext, který by měl být přiřazen přihlašujícímu se uživateli, zkusí použít zde definovaný kontext.

removable_context
Kontext, který bude přiřazen vyměnitelným zařízením.

initrc_context
Kontext, se kterým budou prováděny initrc skripty.

netfilter_contexts
Nastavení kontextů pro příchozí pakety podle použitého protokolu a příchozího portu.

userhelper_context
Kontext pro programy spuštěné pod superuživatelem.

2.1.3 /etc/selinux/(targeted|strict|mls)/contexts/files/

file_contexts
Soubor obsahující stovky cest k souborům v podobě regulárních výrazů a kontext jim přiřazený. Tento soubor je použit při značkování systému. Soubor je sice editovatelný, ale při aktualizaci politiky se tento soubor přepíše.

file_contexts.homedirs
Soubor použitý při značkování domovských adrešářů, vygenerovaný pomocí programu `genhomedircon` z *homedir_template*. Kontexty v něm obsažené jsou rozděleny na SELinux identity (podle mapování v *seusers*).

homedir_template
Šablona pro `genhomedircon` určující, jak mají být označkovány soubory v domovských adresářích.

media
Kontexty pro cdrom, floppy a pevné disky.

/etc/selinux/(targeted|strict|mls)/modules/

Tento adresář obsahuje dva podadresáře *active* a *previous*. V active jsou uloženy soubory (*seusers*, *file_contexts*, *homedir_template* a další), podle kterých je sestavena právě běžící politika a v adresáři modules jsou moduly zavedené do jádra. V previous jsou soubory pro předchozí politiku.

/etc/selinux/(targeted|strict|mls)/policy/

Adresář, kde je uložena politika, která se při startu systému zavádí do jádra.

2.2 /selinux/

Zde je nejdůležitějším adresářem *booleans*, kde jsou konfigurační soubory zabezpečených prvků systému. Booleans je mechanizmus, jak za běhu zapnout/vypnout nějakou službu, aniž by musela být politika jakkoliv upravována.

Například *httpd_enable_cgi*, je implicitně nastaven, jako pravda (1, on). Tímto umožníme uživateli spouštějícímu cgi skript přejít do nové domény, získat správný kontext a další oprávnění. Pokud bychom chtěli spouštění cgi skriptů zakázat, nastavili bychom *httpd_enable_cgi* na false (0, off). Každý z těchto souborů by měl obsahovat právě dvě nuly nebo jedničky. První hodnota určuje *momentální stav*, druhá hodnota je *čekající*, změna na tuto hodnotu se provede až po potvrzení. Jsou dva způsoby, jak booleans nastavovat: pomocí `setsebool` nebo ručně. Příkazem

```
[root@noutec ~]# setsebool httpd_enable_cgi 1
```

změníme obě čísla na 1, tudíž nastavení se ihned provede.

Ruční konfigurace se provádí unixovým příkazem `echo`, pro lepší představu berme za to, že soubor obsahuje dvě nuly.

```
[root@noutec ~]# echo 1 > /selinux/booleans/httpd_enable_cgi
```

Změní obsah *httpd_enable_cgi* na hodnoty *0 1*, to znamená, že tato boolean je stále vypnutá, čeká na potvrzení. Pokud takto nastavíme několik různých booleans, jednotně provedeme jejich aktivaci opět pomocí `echo`

```
[root@noutec ~]# echo 1 > /selinux/commit_pending_bools
```

Abychom přesně věděli, co boolean povoluje, můžeme se podívat do zdrojových souborů, konkrétně *.if* souborů části systému, která nás zajímá (zde apache.if), kde hledáme část, která začíná

```
tunable_policy(`jméno_boolean',`
    # zde jsou pravidla
')
```

2.3 /usr/share/selinux/

Zde jsou v adresářích *strict*, *targeted* a *mls* uloženy všechny dostupné základní moduly (*.pp) pro jednotlivé politiky. Adresář *devel* obsahuje soubory potřebné pro kompilaci vlastních pravidel. V devel je i demonstrační příklad (*example.*[*te*, *fc*, *if*]), jak by mohla vypadat pravidla. Pro konfiguraci vlastních pravidel bude nejlepší použít tento adresář (devel), ve kterém si můžeme vytvořit adresář *local* pro identifikaci, že se jedná o lokální úprávy a v něm vytvořit podadresáře pojmenované podle částí systému, které právě konfigurujeme. V devel najdeme i samotný *Makefile*, který nabízí následující operace:

- load – zkompiluje, vytvoří a nahraje do jádra novou politiku
- reload – zkompiluje, vytvoří novou politiku a pokud byla politika změněna, tak ji zavede do jádra
- clean – smaže pomocný tmp adresář a .pp moduly
- bez parametru – zkompiluje zdrojové soubory v aktuálním adresáři

Při psaní pravidel budeme používat celkem tři soubory s příponami *.fc*, *.if* a *.te*.
V souboru *.fc* s následujícím obsahem se udává, jak mají být označkovány soubory s novým kontextem:

```
cesta  [prepinac]  gen_context(novy_kontext, level)
```

Cesta může být zadána absolutně (soubor) nebo formou regulárního výrazu (adresář). Pokud přepínač vynecháme, budou označkovány všechny soubory a adresáře, s přepínačem -- se označkují pouze soubory. S přepínačem *-d* se označkují pouze adresáře (soubory v takovém adresáři si ponechají svůj kontext, ale nově vytvořené soubory zdědí kontext podle nadřazeného adresáře).

Do souboru *.if* si můžeme nadefinovat vlastní makra. Makra jsou napsána pomocí *makro procesoru m4* a slouží ke zpřehlednění pravidel. Nadefinujeme například makro, které umožnuje aplikaci, kterou konfigurujeme, číst logy. Spousta m4 maker je již přednastavena, přehled některých najdeme v adresáři */usr/share/selinux/devel/include/* v *.spt* souborech, několik zajímavých maker je v příloze.
Například přednastavená makra vypadájí:

```
# makro all_file_perms
define(‘all_file_perms’,‘{ ioctl read write create
getattr setattr lock relabelfrom relabelto append
unlink link rename execute swapon quotaon mounton
execute_no_trans entrypoint execmod }’)

# pravidlo
allow typ1 typ2: file all_file_perms;
```

Abychom nemuseli vypisovat v pravidle všechna výše vypsaná oprávnění, použijeme makro *all_file_perms*, které se při překladu nahradí požadovaným (tzn. všechny operace, které lze se soubory provádět). Makra lze i zamořovat a definovat je pomocí jiných maker. Použití *.if* souboru není povinné a záleží na administrátorovi, jestli do něj vlastní makra nadefinuje. Ta se definují pomocí direktivy *interface*

```
# interface(‘název_makra’,‘
interface(‘povol_spustit’,‘
     gen_require(‘
         # zde nadefinujeme typy použité v makru
         type bin_t;
     ’)
     # zde píšeme pravidla

     allow $1 bin_t: file { read getattr execute };
’)
```

Až zavoláme naše makro *povol_spustit(user_t)*, všechny výskyty `$1` se v makru nahradí typem *user_t*. Počet částí začínající `$` udává počet parametrů, které makro očekává. Při konfiguraci si v podstatě vystačíme pouze se soubory *.fc* a *.te*.

Do souboru *.te* budeme psát samotná *TE pravidla* SELinuxu. Zde můžeme při psaní pravidel použít přednastavená makra a naše makra v *.if* souboru. Obsah souboru by měl vypadat následovně:

```
policy_module(jmeno,verze); # nebo module jmeno verze;

require{
# tuto sekci lze přirovnat k deklaraci proměnných
# definují se zde použité třídy objektů, atributy a \
  typy, které už jsou v systému známy

# atribut pro proces
atrribute domain;

# typ jmeno;
type bin_t, user_t;
```

```
# class třída { množina_operací_dané_třídy };
class file { read getattr execute };
};

# deklarace našich nových typů
# typ, který bude přiřazen procesu
type můj_typ_t, domain;

# naše pravidla
# povol doméně s typem můj_typ_t spustit nebo číst \
  soubory s typem bin_t (soubory v /bin/)
allow můj_typ_t bin_t: file { read getattr execute };

# makro(parametry)
# po spuštění souboru s typem bin_t povol přechod \
  z domény user_t do domény můj_typ_t
domain_auto_trans(user_t, bin_t, můj_typ_t)
```

2.4 Implicitní nastavení ve Fedoře

V základní konfiguraci Fedora používá kontexty *user_u:user_r:user_t*, *staff_u:staff_r:staff_t* pro uživatele, kontext *root:sysadm_r:sysadm_t* pro superuživatele a kontexty *system_u:system_r:** a *system_u:object_r:** pro systémové procesy a soubory.

SELinux identity:

- user_u – identita pro běžné uživatele
- staff_u – identita pro administrativní uživatele
- system_u – identita pro systémové procesy a soubory
- root – identita superuživatele

SELinux role:

- user_r – je role pro běžné uživatele bez možnosti provádět jakékoliv administrativní operace. Jejich bezpečnostní kontext je zpravidla *user_u:user_r:user_t* (striktní politika), se kterým mohou spouštět běžné aplikace a mají přístup ke svým domovským adresářům.
- staff_r – je role pro uživatele, kteří mají v podstatě stejná oprávnění jako běžní uživatelé, ale mohou z ní přejít do role *sysadm_r*, což je role pro administrativní účely
- sysadm_r – role pro administrativní operace
- secadm_r – role pro konfiguraci zabezpečení systému, může spouštět všechny SELinux nástroje, nesmí instalovat software a provádět jiné administrativní operace

- auditadm_r – role pro přístup k logům auditovacího daemona, může měnit nastavení auditovacího subsystému
- system_r – role pro systémové procesy
- object_r – role používaná pro soubory

Ve striktní police se *secadm_r* a *auditadm_r* role nepoužívají. Operace, které jsou oprávněny tyto role provádět, přísluší *sysadm_r* roli. Tyto role jsou uplatněny až v MLS politice a jejich oprávnění jsou roli *sysadm_r* odebrány.

V targeted politice dostává uživatel kontext *user_u:system_r:uncofined_t* a může přistupovat ke všem typům. Doména *unconfined_t* je vytvořena jako alias množiny typů *sysadm_t*, *secadm_t*, *sshd_t* a *auditadm_t*.

Změny rolí
Jestliže zachováme implicitní kontexty, to znamená, že *user_u* je v roli *user_r* a *staff_u* v roli *staff_r*, jsou povoleny následující změny rolí (ve striktní politice).

- user_r – není povolena žádná změna role
- staff_r – povolen přechod do *sysadm_r*
- system_r – není povolena žádná změna role
- root – povoleny přechody do *system_r*, *sysadm_r* a *staff_r*

Přístup role k typům
Subjekty v následujících rolích mají přístup k vypsaným typům. Kompletní přehled typů pro role je pouze pro role *user_r* a *staff_r*. Typů pro roli *sysadm_r* je kolem stovky, pro *system_r* asi 250 typů, proto jejich přehled není kompletní.

user_r
chfn_t, consoletype_t, dhcpc_t, httpd_user_script_t, ifconfig_t, postfix_postdrop_t, pppd_t, loadkeys_t, newrole_t, pam_t, passwd_t, ping_t, system_chkpwd_t, traceroute_t, user_crond_t, user_cdrecord_t, user_chkpwd_t, user_crontab_t, user_t, user_tvtime_t, user_dbusd_t, insmod_t, user_gpg_helper_t, user_evolution_webcal_t, user_games_t, user_gconfd_t, user_gpg_agent_t, user_gpg_pinentry_t, user_gpg_t, user_iceauth_t, user_irc_t, user_javaplugin_t, user_lockdev_t, user_lpr_t, user_mail_t, user_mencoder_t, user_mozilla_t, user_mplayer_t, user_screen_t, user_spamassassin_t, user_spamc_t, user_ssh_agent_t, user_ssh_keysign_t, user_ssh_t, user_su_t, user_sudo_t, user_thunderbird_t, user_uml_t, user_userhelper_t, user_xauth_t, user_xserver_t, usernetctl_t, utempter_t, user_ethereal_t, user_evolution_t, user_evolution_alarm_t, user_evolution_exchange_t, user_evolution_server_t

staff_r
stejné typy jako *user_r*, pouze místo prefixu *user* mají prefix *staff*

sysadm_r
výše popsané typy, s prefixem *sysadm*, a navíc typy jako bootloader_t, checkpolicy_t, depmod_t, load_policy_t, mount_t, rpm_t, rpm_script_t, semanage_t, setfiles_t a další

system_r
alsa_t, apm_t, auditd_t, bootloader_t, cupsd_t, getty_t, httpd_t, httpd_ a dalších asi 240 typů

Přístup typů k typům
V TE pravidlech jsou povoleny přístupy *user_t* (resp. *staff_t*, *sysadm_t*) k vypsaným typům. Opět se nejedná o kompletní výpis, ale jen o přehled.

user_t
agp_device_t, alsa_etc_rw_t, apmd_t, bin_t, bluetooth_t, console_device_t, cupsd_t (a různé s prefixem cupsd), device_t, evolution_∗, games_exec_t, init_t, httpd_∗, ld_so_t, lib_t, locale_t, sshd_t, shlib_t, sound_device_t, ssh_exec_t, su_exec_t, user_∗ (soubory v domovských adresářích), var_t, xdm_t (a různé s prefixem var a xdm), usr_t a spousta dalších typů

staff_t
stejné typy jako u *user_t* (prefix *staff*)

sysadm_t
typ *sysadm_t* má přístup i ke všem ostatním typům

Kapitola 3

Základní příkazy a nástroje

Prozatím jsme si řekli jak co funguje, ale ještě nevíme, jak pozměnit nebo zjistit základní nastavení SELinuxu v systému, jak změnit politiku, přeznačkovat soubory a jak naše pravidla uvést v chod. Tato kapitola obsahuje i souhrn některých dostupných nástrojů ke konfiguraci částí SELinuxu.

3.1 Změna politiky

Změnit politiku lze dvěma způsoby: buď v grafickém módu, příkazem `system-config-selinux`, kde se dají nastavit i další části (mapování uživatelů, změny rolí, přehled typů a další), nebo v terminálu. Tento postup si popíšeme – editujeme */etc/selinux/config* (zkráceně *config*) a změníme *SELINUXTYPE* na požadovanou politiku a *SELINUX* na *permissive*. Politiku nelze změnit za běhu systému, musíme restartovat počítač. Potřebujeme však ještě vynutit přeznačkování celého systému pomocí:

```
[root@noutec ~]# touch /.autorelabel
```

Nyní můžeme restartovat. Po přeznačkování a znovunaběhnutí opět editujeme *config* a nastavíme *SELINUX* na *enforcing*.

3.2 Změna módu

Změna módu podle konfiguračního souboru se provede až při následujícím startu systému, což by bylo poněkud nekomfortní. Proto narozdíl od změny politiky, mód lze měnit za běhu příkazem `setenforce` s parametrem `0` (vypne enforcing) nebo `1` (zapne enforcing), čehož hojně využijeme při konfiguraci systému. V předchozím příkladě jsme sice editovali *config*, ale ke skutečné změně ještě nedošlo. Změnu tedy provedeme:

```
[root@noutec ~]# setenforce 1
```

3.3 Kompilace pravidel a zavedení do jádra

Až napíšeme nějaká pravidla, musíme je přeložit a zavést do jádra. Kompilaci provedeme v adresáři s pravidly příkazem:

```
[root@noutec ~]# make -f /usr/share/selinux/devel/Makefile
```

Pravidla se zkompilují, ale nezavedou se do jádra (viz. parametry v 2. kapitole). Ruční zavedení (i odstranění, výpis modulů v jádře a další) politiky (.pp) do jádra se provádí příkazem `semodule`.

Zavedení modulu *náš_modul.pp* do jádra:

```
[root@noutec ~]# semodule -i náš_modul.pp
```

Odstranění modulu *náš_modul* z jádra:

```
[root@noutec ~]# semodule -r náš_modul
```

Výpis všech modulů zavedených v jádře:

```
[root@noutec ~]# semodule -l
```

Jestliže zavedeme modul do jádra, bude při každém startu systému *automaticky* nahráván.

3.4 Přeznačkování souborů

Jestliže vytvoříme nový kontext pro soubory nebo adresáře v domovských adresářích, aktualizuje se kontextový soubor *homedir_template*, ale změna se prozatím neprojevila v souboru *file_contexts.homedirs*, podle kterého se už soubory nebo adresáře značkují. Aktualizaci provedeme příkazem `genhomedircon`. Následně lze již přeznačkovat domovské adresáře novým kontextem. K tomu slouží příkaz `fixfiles`, který dostane jako parametr `relabel`. Pokud spustíme

```
[root@noutec ~]# fixfiles relabel
```

přeznačkuje se celý systém, což je v tomto případě zbytečné. Vystačíme si s

```
[root@noutec ~]# fixfiles relabel /home/
```

a přeznačkuje se pouze adresář */home/* nebo jiný adresář, který dostane `fixfiles` za parametr. Když budeme chtít přeznačkovat pouze jeden soubor, například spustitelný program, který jsme nadefinovali jako vstupní bod do nové domény, vystačíme i s příkazem

```
[root@noutec ~]# restorecon /usr/bin/firefox
```

Změnu role provedem příkazem `newrole`

```
[honza@noutec ~]$ newrole -r nová_role
```

a autentizujeme se svým přihlašovacím heslem.

Příkazem `sestatus` zjistíme informace o SELinuxu, zejména mód a aktivní poli-

tiku. Dostupných nástrojů je opravdu mnoho, některé si už jen vypíšeme (popis viz. manuálové stránky): `seinfo`, `setfiles`, `getsebool`, `setsebool`, `semanage`, `chcat`, `chcon`, `setenforce`, `getenforce`, `checkmodule`, `checkpolicy`, `restorecon` (daemon `restorecond` zajišťuje správné označkování nových souborů), `secon`, `sechecker`, `genhomedircon`, `semodule`, `audi2allow` (projde logy a vytvoří pravidla pro všechny zamítnuté operace), `audit2why` (projde logy a vypíše důvody, proč operace nebyly povoleny) a další.

Novinkou ve Fedora Core 6 je sada GUI nástrojů, dostupných v balíčcích *setools*, *setools-gui* a *setroubleshoot*. Pomocí těchto nástrojů můžeme porovnávat politiky, analyzovat politiky (jejích pravidla, použité typy a role a další informace) a procházet logy. A již dříve zmiňovaný `system-config-selinux` pro konfiguraci politiky, identit, rolí, změn rolí, MLS překladu, nahrávání/odstranění modulů do/z jádra a nastavování boolean.

Kapitola 4

Příklady úprav pravidel

Po, doufejme, zdárném pochopení všech principů si ukážeme, jak nové poznatky uplatnit v praxi. Od základního vytvoření typů a rolí, povolení přístupů a přechodů, po složitější příklad s konfigurací internetového prohlížeče Firefox, kde si ukážeme, kam a po čem sáhnout, pokud nám nebude něco fungovat, tak jak má.

4.1 Mapování uživatelů

Pokud přidáme nového uživatele, je implicitně mapován na SELinux uživatele *user_u* (viz. seusers), tedy základní identita pro neadministrativní uživatele. Pokud bychom chtěli uživateli povolit upravovat systém, tak mu musíme povolit přejít do role, která je k tomu oprávněná, tedy role *sysadm_r*, tento přechod můžeme udělat z role *staff_r*.

```
[root@noutec ~]# semanage login -a -s staff_u honza
```

Kontext uživatele honza je teď *staff_u:staff_r:staff_t*, ale soubory v domovských adresářích jsou označkované stále pro *user_u:user_r:user_t*. Provedeme tedy přeznačkování domovského adresáře podle *file_contexts.homedirs*.

```
[root@noutec ~]# fixfiles relabel /home/honza/
```

4.2 Definice nového typu/domény

I přes velké množství již definovaných typů je pravděpodobné, že budeme potřebovat náš vlastní nový typ. Definujeme typ/doménu pro klienta `pidgin` (dříve gaim). Použijeme *pidgin_t* jako doménu pro proces, *pidgin_exec_t* pro spustitelný soubor, pomocí kterého přejdeme do domény *pidgin_t* a *user_pidgin_home_t* pro soubory pidginu v domovských adresářích. Vytvoříme soubory *pidgin.fc* a *pidgin.te*.

```
# soubor pidgin.te
# nový typ (doména) pidgin_t s atributem domain
type pidgin_t, domain;
# nový typ pidgin_exec_t pro spustitelný soubor
type pidgin_exec_t, exec_type, file_type;
# nový typ pro soubory v domovském adresáři
```

```
type user_pidgin_home_t, file_type;

# soubor pidgin.fc
# spustitelný soubor pro přechod do domény
/usr/bin/pidgin   -- \
    gen_context(system_u:object_r:pidgin_exec_t, s0)
# označkování domovských adresářů
/HOME_DIR/\.purple(/.*)?  \
    gen_context(user_u:object_r:user_purple_home_t, s0)
```

4.3 Přechodová pravidla

Navážeme na předchozí příklad a povolíme programu s typem *pidgin_exec_t*, spuštěnému uživatelem s typem *user_t*, přejít do domény *pidgin_t*.

```
domain_auto_trans(user_t, pidgin_exec_t, pidgin_t)
allow pidgin_t pidgin_exec_t: file { entrypoint execute read \
                                     getattr };
```

4.4 Nová role

Nadefinujeme novou roli *player_r*. To sebou ovšem přináší několik problémů – musíme povolit některým identitám (SELinux uživatelům) se do této role přepnout, musíme vytvořit novou doménu (např. player_t) pro *player_r*, povolit roli přístup k typu *player_t* a upravit konfigurační soubor *default_type* (viz. část 2.1.2)

```
# definujeme novou roli player_r
role player_r;
# vytvoříme jí doménu player_t
type player_t, domain;

# user identita roles { množina_typů } \
  level lvl range rng;
# povolíme identitě user_u vystupovat v roli \
  player_r, level s0 udává oprávnění a range \
  kategorii pro roli player_r
user user_u roles player_r level s0 range s0;

# povolíme roli player_r přistupovat k typům player_t
role player_r types { player_t };


# editujeme soubor default_type a připíšeme řádek
# po příkazu newrole -r player_r bude uživateli \
```

```
  přiřazen kontext user_u:player_r:player_t
player_r:player_t
```

Po zkompilování a zavedení modulu do jádra se v systému objevila nová role *player_r*, která zatím nemá povolen přístup k jinému typu než *player_t*. Musíme tedy přidat další typy, ke kterým budeme moci v roli *player_r* přistupovat.

4.5 Přístup role k typům

Nově vytvořené (z předchozího příkladu) nebo již definované roli přiřadíme typy, ke kterým bude moci přistupovat.

```
# role naše_role types { množina_typů };
role player_r types { games_t };
```

4.6 Přístup domény k typu

Tato pravidla patří mezi ty zásadní a v podstatě skoro všechna pravidla, která napíšeme budou takto vypadat.

```
# akce vstupní_typ cílový_typ: třída { operace };
allow sshd_t sshd_tmp_t: file { read getattr setattr \
                                create write rename };
```

Toto pravidlo povolí doméně *sshd_t* vytvářet, číst, zapisovat a přejmenovat soubory s typem *sshd_tmp_t*.

4.7 Konfigurace prohlížeče Firefox

Idea je taková: zamezíme Firefoxu veškerý přístup do systému, až na ten, který je nezbytně nutný pro samotnou funkci prohlížeče. Při konfiguraci je použita *striktní politika v permissive módu* (pro připomenutí: SELinux je vypnutý, ale prováděné operace jsou logovány). Použijeme audit daemona *auditd* a modul *enabledaudit.pp*. SELinux politika má nastaveno, aby některé AVC zprávy (převážně dontaudit pravidla) nebyly logovány. S tímto modulem se logy budou velice rychle a opravdu vším prováděným plnit. V logu budeme vyhledávat AVC zprávy (AVC msg) a podle jejich obsahu budeme konfigurovat pravidla.

Začneme tím, že konfigurační soubory v domovských adresářích oddělíme (typem) od ostatních. Vytvoříme tedy nový typ pouze pro Firefox – typ *user_firefox_home_t* a tímto typem označkujeme *∼/.mozilla* adresáře. V adresáři */usr/share/selinux/devel/* vytvoříme adresář *local* a v něm podadresář *firefox*, kam umístíme soubory *firefox.te* a *firefox.fc*.

```
# soubor firefox.fc
HOME_DIR/\.mozilla(/.*)?  \
    gen_context(user_u:object_r:user_firefox_home_t, s0)
```

Teď je potřeba vytvořit nový typ (doménu) pro samotný Firefox, se kterým bude moci k těmto souborům přistupovat a označkovat spouštěcí soubor, pomocí kterého Firefox do této domény přejde. Do *firefox.fc* přidáme:

```
# vstupní bod do domény
/usr/bin/firefox  --  \
    gen_context(system_u:object_r:firefox_exec_t, s0)
```

Nadále vytvoříme typ *user_download_home_t* pro adresář ∼/*download*, kde bude mít doména *firefox_t* povoleno číst a ukládat data.

```
# adresář pro ukládání stažených souborů
HOME_DIR/download(/.*)?  \
    gen_context(user_u:object_r:user_download_home_t, s0)
```

Více už do souboru *firefox.fc* připisovat nebudeme. Nyní vytvoříme soubor *firefox.te* a v něm definujeme nové typy, které budeme potřebovat (všechna následující pravidla už píšeme do *firefox.te*).

```
policy_module(firefox, 1.0.0);
require{
attribute domain;    # atribut pro doménu
attribute exec_type; # atribut pro spustitelný soubor
attribute file_type; # atribut pro soubory, adresáře
};
# část s pravidly

# typ pro doménu(atribut domain)
type firefox_t, domain;

# typ pro spustitelný (exec_type) soubor na \
  disku (file_type), jenž použijeme jako vstupní bod
type firefox_exec_t, exec_type, file_type;

# typ pro soubory v domovském adresáři (atribut file_type)
type user_firefox_home_t, file_type;
# typ pro soubory v adresáři download
type user_download_home_t, file_type;
```

Zkompilujeme, zavedeme do jádra (viz. část 3.3) a přeznačkujeme soubory v domovských adresářích a spustitelný soubor:

```
[root@noutec ~]# fixfiles relabel /home/ /usr/bin/firefox
```

Pokud nyní jako běžný uživatel vyzkoušíme vypsat atributy adresáře ∼/download nebo ∼/.mozilla, dostaneme následující výsledek (v enforcing módu):

```
[honza@noutec ~]$ ls -l download
?--------- ? ?     ?        ? download
```

To znamená, že ačkoliv jsou soubory v našem domovském adresáři, nemáme povoleno je číst a vypisovat atributy. Připíšeme následující (podobná pravidla můžeme aplikovat na ∼/.mozilla). Poté zkompilujeme, zavedeme do jádra a ověříme správné chování.

```
# do části require
type user_t;
class file { create_file_perms };
class dir { rw_dir_perms };

# část s pravidly
allow user_t user_download_home_t: dir rw_dir_perms;
allow user_t user_download_home_t: file create_file_perms;
```

Poznámka Pro zpřehlednění již nebudeme v následujících výpisech psát část *require*. Pokud budeme pracovat s nějakým typem, který jsme ještě nepoužili, tak ho v části *require* zapíšeme. To stejné platí i pro třídu a množinu operací nad danou třídou. Budeme tedy psát pouze TE pravidla.
Obdobná oprávnění musíme udělit i doméně *firefox_t*. Aby se doména dostala až k adresáři *∼/.mozilla*, musí mít oprávnění projít celou cestu k tomuto adresáři. Adresář */home/* má typ *home_root_t*, doména musí mít povoleno prohledávat tento adresář a získat atributy adresářů. Stejný důvod, pouze v */home/* o úroveň níže, má druhé pravidlo.

```
allow firefox_t home_root_t: dir { search getattr };
allow firefox_t user_home_dir_t: dir { search getattr };
```

Teď můžeme napsat samotná pravidla pro přístup do *∼/.mozilla*, kde bude moci doména *firefox_t* číst, zapisovat, vytvářet nové soubory a adresáře (například pro ukládání do cache, bookmarků a nastavení).

```
# doména firefox_t může číst, vytvářet, přejmenovat (rename), \
  uzamknout (lock), zapisovat nebo zrušit odkaz na soubor \
  s typem user_firefox_home_t
allow firefox_t user_firefox_home_t: file { getattr read write  \
                                     create rename unlink lock };
# doména firefox_t může prohledávat, číst, přidávat/odebírat
  položky z adresáře s typem user_firefox_home_t
allow firefox_t user_firefox_home_t: dir { search getattr write \
                                     add_name remove_name read };
```

Dostáváme se k části, jak bezpečně změnit doménu na *firefox_t*. Napíšeme pravidla pro přechod z *user_t*, přes *firefox_exec_t* do domény *firefox_t*. Buď použijeme přednastavené makro *domain_auto_trans*, a nebo si pravidla celé napíšeme sami. Nejprve nadefinujeme samotný přechod (ten ale nic nepovoluje). Pak povolíme typu *user_t* spustit soubor pro vstup do domény a povolíme přechod z *user_t* do *firefox_t*. Čtvrté pravidlo povolí typu *firefox_t* použít typ *firefox_exec_t* jako vstupní bod do domény. Pro čtení souboru jsou nutná oprávnění *read* a *getattr*. Ve striktní politice se rozhoduje i na základě role, proto musíme navíc povolit roli *user_r* přístup k typu *firefox_t*.

```
# pokud user_t spustí soubor s typem firefox_exec_t vznikne \
  proces v doméně firefox_t
type_transition user_t firefox_exec_t: process firefox_t;

# user_t může spustit soubor s typem firefox_exec_t a použít ho \
  pro přechod do jiné domény
allow user_t firefox_exec_t: file { execute read getattr \
                                    entrypoint }
# user_t smí přejít do domény firefox_t
allow user_t firefox_t: process transition;

# pokud je spustěn soubor s typem firefox_exec_t, je povolen
  přechod do domény firefox_t
allow firefox_t firefox_exec_t: file { entrypoint read getattr \
                                       execute };

# pomoci makra domain_auto_trans by pravidla vypadala následovně
domain_auto_trans(user_t, firefox_exec_t, firefox_t)
allow firefox_t firefox_exec_t: file { execute entrypoint read \
                                       getattr };

# přístup role k typu
role user_r types { firefox_t };
```

Předchozí pravidla je možné napsat celkem intuitivně, nyní se dostáváme do části, kde přijdou na řadu logovací soubory. Zavedeme do jádra modul *enabledaudit.pp*.

```
[root@noutec ~]# semodule -b /usr/share/selinux/\
strict/enableaudit.pp
```

Zkusíme spustit prohlížeč `firefox`, běžící již v doméně *firefox_t*, prohlédneme logovací soubor ve */var/log/audit/* a budeme hledat záznamy jako tento:

```
type=AVC msg=audit(1178479954.199:12870):
avc:  denied { execute_no_trans } for  pid=31239 comm="firefox"
name="firefox" dev=hda7 ino=1205324
scontext=user_u:user_r:firefox_t:s0
tcontext=system_u:object_r:firefox_exec_t:s0 tclass=file
```

Kde jsou informace o zakázané akci. Nejdůležitějšími částmi jsou pro nás *scontext*, *tcontext*, *tclass* a *denied { operace }*. AVC zpráva popisuje nepovolený pokus domény *firefox_t* (scontext) o *spuštění bez změny typu* (execute_no_trans) *souboru* (tclass=file) s typem *firefox_exec_t* (tcontext). Další pravidla se budou odvíjet na základě AVC zpráv.
Upravíme tedy poslední pravidlo na

```
allow firefox_t firefox_exec_t: file { entrypoint read getattr \
                                       execute execute_no_trans };
```

Pravidlem

```
allow firefox_t firefox_t: shm { create unix_read unix_write \
                                 read write destroy };
```

povolíme doméně *firefox_t* vytvořit, číst, zapisovat a odstranit sdílenou paměť.
Přístup k síti zajistí následující pravidla

```
# povolení vytvořit si, číst, zjistit parametry, připojit se
  a zapisovat do tcp_socketu a udp_socketu
allow firefox_t firefox_t: tcp_socket { create connect getopt \
                                    read write name_connect };
allow firefox_t firefox_t: udp_socket { create connect getattr \
                                            read write ioctl};
# povolení pojmenovat tcp_socket a posílat a přijímat zprávy
allow firefox_t http_port_t: tcp_socket { name_connect send_msg \
                                          recv_msg};
# povolení posílat/přijímat udp a tcp pakety sít'ové kartě
allow firefox_t netif_t: netif { udp_send udp_recv tcp_send \
                                 tcp_recv };
allow firefox_t node_t: node { udp_send udp_recv tcp_send \
                               tcp_recv };
# doména firefox_t smí posílat/přijímat zprávy od udp_socket
  s typem dns_port_t
allow firefox_t dns_port_t: udp_socket { send_msg recv_msg };
# čtení nastavení sítě
allow firefox_t net_conf_t: file { read getattr };
# self reprezentuje doménu firefox_t
allow firefox_t self: netlink_route_socket { create bind getattr \
                                             write nlmsg_read read};
```

Povolíme doméně *firefox_t* číst a spouštět sdílené knihovny. Typy *ld_so_t*, *shlib_t*, *textrel_shlib_t* a *lib_t* jsou označkovány sdílené knihovny. Povolíme číst a spouštět soubory s těmito typy, přečíst odkaz a procházet a číst adresář s typem *lib_t*

```
allow firefox_t ld_so_t: file { read execute getattr };
allow firefox_t ld_so_cache_t: file { read getattr };
```

```
allow firefox_t shlib_t: file { read getattr execute };
allow firefox_t textrel_shlib_t: file { read getattr execute };
allow firefox_t lib_t: dir { search read getattr };
allow firefox_t lib_t: lnk_file { read getattr };
allow firefox_t lib_t: file { read getattr execute \
                              execute_no_trans };
```

Firefox_t doména si může zjistit (*getsched*) a změnit (*setsched*) svou prioritu procesu a posílat si signály.

```
allow firefox_t firefox_t: process { getsched setsched \
                                     sigkill signal };
```

Přístup domény prohledávat adresář, číst a zapisovat do dočasných souborů okenního správce (*xdm_tmp_t*), povolení číst nastavení lokalizace (*locale_t*) a fonty (*fonts_t*).

```
allow firefox_t xdm_tmp_t: dir { search };
allow firefox_t xdm_tmp_t: file { write read getattr };
allow firefox_t xdm_tmp_t: sock_file { write };
allow firefox_t fonts_t: dir { getattr search };
allow firefox_t fonts_t: file { read getattr };
allow firefox_t locale_t: dir { search };
allow firefox_t locale_t: file { read getattr };
```

Pravidel je celkem hodně a nebudeme je zde všechny vypisovat. Postupnou analýzou AVC zpráv budou pravidla přibývat, není však účelem povolit veškeré zakázané operace, protože by se mezi nimi mohly skrývat i akce, které rozhodně nejsou potřebné nebo žádoucí. Je tedy na administrátorovi se nad konfigurací zamyslet a ne pouze připisovat pravidla a všechno povolit.

Místo pročítání logů, můžeme použít nástroj `audit2allow`, který projde log a vygeneruje z něj *.te* soubor, ve kterém povolí všechny zakázané operace a my jen odstraníme pravidla, která nechceme.

Kapitola 5

Závěr

V této práci jsem kladl za cíl podat jasnou představu o principech SELinuxu, jeho význam a uplatnění v linuxových systémech. Pokud se nám podaří řádně tyto problémy pochopit, neměla by jejich implementace ve skutečném světě již nikomu dělat potíže. Od jednoduchých příkladů jsme se dostali ke konfiguraci internetového prohlížeče Firefox. Na části pravidel jsme si ukázali, jak Firefoxu omezit přístup do systému. Povolili jsme mu přístup do uzavřené části systému. Pokud by došlo k útoku na systém skrze prohlížeč, útočník získá se jen velmi omezený přístup k systému.

I když SELinux poskytuje nezanedbatelné zlepšení bezpečnosti systému, na kterém běží, je prozatím vhodný spíše pro použití na serverech, než na pracovních stanicích. Všechny klady u spousty uživatelů přebijí dva jednoduché fakty – čas strávený studiem principů a nad konfigurací. Přitom principy SELinuxu jsou založeny na celkem jednoduchých myšlenkách. Samotná konfigurace už je jen vyústěním našich nových znalostí.

V budoucnu by mohla být dokumentace směřována o něco hlouběji. Nyní jsme se věnovali částem, které byly zaměřeny na uživatelský prostor. Rozšířením by mohla být studie zaměřující se na implementaci v samotném jádře systému.

Kapitola 6

Přílohy

V této části jsou shrnuty třídy objektů, atributy typů, operace nad objekty, některá makra a SELinux typy pro soubory. U atributů, maker a typů se jedná pouze o zlomek, jelikož kupříkladu typů je více než tisíc, a není cílem je zde všechny popisovat.

6.1 SELinux typy

Seznam opravdu zlomku typů, které SELinux používá. V přístupových pravidlech se používají na pozicích

allow typ1 **typ2**: třída { operace };

Typ	Typ pro
bin_t	soubory v /bin/
default_t	implicitní kontext
etc_runtime_t	měnící se soubory v /etc/ a podadresářích
etc_t	neměnící se soubory v /etc/ a podadresářích
file_t	pro neoznačkované soubory
lib_t	moduly, knihovny a jiné soubory s nimi spjaté soubory v /lib/
readable_t	soubory a adresáře čitelné běžnými uživateli
root_t	kořenový filesystem
sambafs_t	samba filesystem
sbin_t	soubory v /sbin/ a /usr/sbin/
shell_exec_t	shell (např. /bin/bash)
shlib_t	sdílené knihovny v /lib/ a /usr/lib/
tmp_t	uživatelské soubory v /tmp/
unlabeled_t	neoznačkové soubory (vytvořené, když byl SELinux vypnut)
user_home_t	soubory v domovských adresářích
usr_t	soubory v /usr/ a /opt/
var_lib_t	soubory ve /var/lib/
var_t	soubory ve /var/

Tabulka 6.1: Typy souborů

Typ	Popis
icmp_socket_t	soket pro posílání ICMP zpráv
igmp_packet_t	IGMP paket
netif_t	síťové rozhraní
node_t	implicitní typ pro síťový uzel
pop_port_t	POP port
port_t	TCP/IP port
scmp_packet_t	SCMP paket
tcp_socket_t	soket pro posílání TCP dat

Tabulka 6.2: Typy pro síťové prvky

Typ	Typ pro
default_context_t	soubory v /etc/selinux/*politika*/context/
file_context_t	soubory v /etc/selinux/*politika*/files/
no_access_t	soubory přístupné pouze administrátorovi
selinux_config_t	soubory v /etc/selinux/

Tabulka 6.3: Typy spojené se SELinuxem

6.2 Třídy objektů

Třída objektů reprezentuje objekty, které jsou podobné a je s nimi i stejně zacházeno (soubor, adresář, odkaz atd.). Třída se používá v přístupových pravidlech.

allow typ1 typ2: **třída** { operace };

Třída	Reprezentuje
blk_file	soubor blokového zařízení
chr_file	soubor znakového zařízení
dir	adresář
fd	filedeskriptor
fifo_file	soubor fronty
file	soubor
filesystem	zformátovaný souborový systém na diskovém oddílu
lnk_file	tvrdý nebo symbolický odkaz
sock_file	soubor síťového soketu

Tabulka 6.4: Třídy souborů

Třída	Reprezentuje
key_socket	IPSec soket
netif	síťové rozhraní
netlink_socket	soket použitý pro komunikaci s jádrem pomocí netlink systémových volání
node	TCP/IP host
raw_socket	raw IP soket
socket	soket
tcp_socket	TCP soket
udp_socket	UDP soket

Tabulka 6.5: Třídy síťových prvků

6.3 SELinux operace

Seznam některých nejdůležitějších operací, které lze provádět nad danými třídami objektů. Operace se používají v přístupových pravidlech

allow typ1 typ2: třída_objektů { **povolené_operace** };

Třídy objektů	Operace	Povoluje
dir chr_file fifo_file file lnk_file blk_file	append, execute, create, link, write, read, rename	přidávat, spouštět, vytvářet, udělat odkaz, zapisovat a číst soubor nebo adresář

Tabulka 6.6: Přehled základních operací se soubory a adresáři

Třídy objektů	Operace	Povoluje
ipc msgq sem shm	create, destroy, read, write	vytvořit, zrušit, číst a zapisovat do IPC fronty, semaforu a sdílené paměti

Tabulka 6.7: Přehled základních operací se sdílenou pamětí a IPC frontami

Třídy objektů	Operace	Povoluje
key_socket udp_socket tcp_socket netlink_socket raw_ipsocket unix_dgram_socket unix_stream_socket socket packet_socket	create, bind, append, read, write, accept	vytvořit, pojmenovat, přidat, číst, zapisovat a přijmout ze soketu

Tabulka 6.8: Přehled základních operací se sokety

Třídy objektů	Operace	Povoluje
dir	add_name	přidávat položky do adresáře
file	entrypoint	vsoupit do nové domény přes tento soubor
capability	chown	změnit vlastníka nebo skupinu
process	fork	fork procesu
dir	rmdir	smazat adresář
dir	search	prohledávat adresář
fd	use	použít filedeskriptor

Tabulka 6.9: Přehled dalších operací

6.4 Atributy typů

Seznam několika atributů pro domény a typy používaných při definici nových typů pomocí type nový_typ, **seznam_atributů**;

Atribut	Popis domény
admin	administrátorská doména (obdoba sysadm_t)
etc_writer	doména, která může zapisovat do etc_t
privlog	doména, která může komunikovat s logovacím daemonem
privmail	doména, která může přejít do system_mail_t domény
privmem	doména, která může přistupovat k paměti jádra
privmodule	doména, která může spouštět modprobe
privrole	doména, která může změnit roli uživatele
privuser	doména, která může změnit identitu uživatele

Tabulka 6.10: Atributy domén

Atribut	Popis typu
domain	typ pro proces
exec_type	spustitelný soubor sloužící jako vstupní bod domény
file_type	typ pro soubory
home_dir_type	typ pro adresáře obsahující domovské adresáře
home_type	typ pro domovské adresáře
logfile	typ pro logovací soubory a adresáře
netif_type	typ pro síťová rozhraní
socket_type	typ pro jádrem vytvořené sokety
sysadmfile	typ pro administrátorem kontrolované soubory
tmpfile	typ pro dočasné soubory
unpriv_userdomain	typ pro běžné uživatele (obdoba user_t)
user_home_type	typ pro domovské adresáře běžných uživatelů

Tabulka 6.11: Atributy typů

6.5 SELinux makra

Seznam maker resprezentujících množinu podobných objektů, soubory oprávnění nad objekty a makra povolující soubor specifických operací jako přechody do domén, přístup k sítí a další. Makra pro třídy se používají v přístupových pravidlech na následující pozici

allow typ1 typ2: **makro** operace ;

Makra pro souborová, soketová oprávnění a IPC makra na pozici

allow typ1 typ2: třída **makro**;

Poslední typ maker se používá místo přístupových pravidel

makro(argumenty)

Makro	Oprávnění
rw_socket_perms	používat soket
create_socket_perms	vytvořit a používat soket
rw_stream_socket_perms	používat stream soket
create_stream_socket_perms	vytvářet a používat stream soket

Tabulka 6.12: Makra pro soketová oprávnění

Makro	Oprávnění
stat_file_perms	volat na soubor stat a číst soubor
x_file_perms	spouštět soubor
r_file_perms	číst soubor
rx_file_perms	číst a spouštět soubor
rw_file_perms	číst a zapisovat do souboru
ra_file_perms	číst a přidávat do souboru
link_file_perms	vytvářet, rušit odkaz a přejmenovávat soubor
create_file_perms	vytvářet, mazat a přistupovat k souboru
r_dir_perms	číst a prohledávat adresář
rw_dir_perms	číst a měnit adresář
ra_dir_perms	číst a přidávat do adresáře
create_dir_perms	vytvářet, mazat a přistupovat k adresáři
mount_fs_perms	připojovat a odpojovat

Tabulka 6.13: Makra pro souborová oprávnění

Makro	Reprezentuje
dir_file_class_set	všechny soubory a adresáře
file_class_set	všechny soubory (bez adresářů)
devfile_class_set	soubory reprezentující zařízení (/dev/*)
notdevfile_class_set	všechny soubory kromě souborů reprezentujících zařízení
socket_class_set	všechny sokety
dgram_socket_class_set	datagramové sokety
stream_socket_class_set	stream sokety
unpriv_socket_class_set	neprivilegované sokety (bez rawip, netlink, packet, key)

Tabulka 6.14: Makra pro třídy

Makro	Oprávnění
r_sem_perms	číst semafor
rw_sem_perms	vytvořit a používat semafor
r_msgq_perms	číst zprávy ve frontě
rw_msgq_perms	vytvářet a používat zprávy ve frontě
r_shm_perms	číst ze sdílené paměti
rw_shm_perms	vytvořit a používat sdílenou pamět

Tabulka 6.15: IPC makra

Makro s parametry	Autorizuje doménu
domain_trans(doména, typ, nová doména)	přejít přes spustitelný soubor do nové domény
domain_auto_trans(doména, typ, nová doména)	automaticky přejít přes spustitelný soubor s typem do nové domény
can_exec(doména, typ)	spustit program s typem bez přechodu do jiné domény

Tabulka 6.16: Makra pro přechod/bez přechodu do jiné domény a spuštění

Literatura

[1] MCCARTY Bill. *SELinux*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2004, 254s. ISBN 0-596-00716-7

[2] *Discretionary Access Control* [online], Last modified 20 April 2007 [cit. 2007-01-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Discretionary_access_control>.

[3] *Mandatory Access Control* [online], Last modified 3 May 2007 [cit. 2007-01-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Mandatory_access_control>.

[4] *Fedora SELinux Project Pages* [online], Poslední změna 2007-03-20 [cit. 2007-01-22]. Dostupný z WWW: <http://fedoraproject.org/wiki/SELinux>.

[5] *Configuring the SELinux Policy* [online]. Dostupný z WWW: <http://www.nsa.gov/selinux/papers/policy2/t1.html#INTRO>.

[6] LOSCOCCO, Peter. Integrating Flexible Support for Security Policies into the Linux Operating System [online], 2001. Dostupný z WWW: <http://www.nsa.gov/selinux/papers/slinux/slinux.html>.

[7] *Red Hat Enterprise Linux 4: Red Hat SELinux Guide* [online], [2005]. Dostupný z WWW: <http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/rhlcommon-chapter-0001.html>.

[8] MORRIS, James. *An Overview of Multilevel Security and LSPP under Linux* [online], 2005 [cit. 2007-03-13]. Dostupný z WWW: <http://james-morris.livejournal.com/5020.html>.

[9] MORRIS, James. *A Brief Introduction to Multi-Category Security (MCS)* [online], 2005 [cit. 2007-03-13]. Dostupný z WWW: <http://james-morris.livejournal.com/5583.html>.

[10] CAPLAN, David, MACMILLAN, Karl, MAYER, Frank. *SELinux Concepts* [online]. 2006 [cit. 2007-04-17]. Dostupný z WWW: <http://www.informit.com/articles/article.asp?p=606586&rl=1>.

[11] COKER, Faye. *Writing SE Linux policy HOWTO* [online], Last update 18 March 2004. Dostupný z WWW: <http://www.lurking-grue.org/writingselinuxpolicyHOWTO.html#conffiles3>.

[12] WALSH, Dan. *danwalsh's Journal* [online]. Dostupný z WWW: <http://danwalsh.livejournal.com>.