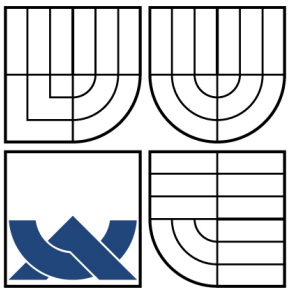
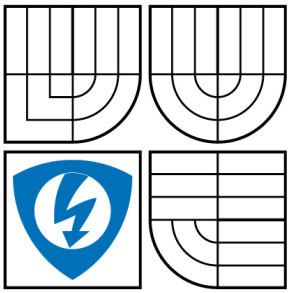


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

MAC ŘÍZENÍ PŘÍSTUPU

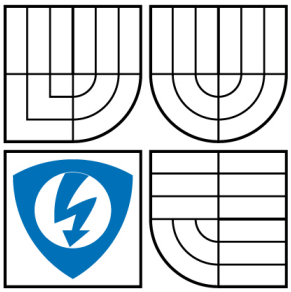
MANDATORY ACCESS CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MIROSLAV GREPL
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ PELKA
SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor

Telekomunikační a informační technika

Student:	Grepl Miroslav Bc.	***ID:***	88923
Ročník:	2	***Akademický rok:***	2007/2008

NÁZEV TÉMATU:

MAC řízení přístupu

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je přehledně v diplomové práci popsat možnosti jedné z implementací MAC řízení přístupu, jež nese název SELinux. Student se v práci soustředí na popis dostupných, standardních pravidel pro nejčastěji provozované serverové služby jakými jsou ssh, ftp, http server. Jednotlivé služby rovněž zabezpečí svými vlastními, na míru upravenými pravidly. Student tyto dvě metody vzájemně porovná a zhodnotí. Dalším úkolem je návrh metodologie tvorby vlastních bezpečnostních pravidel.

DOPORUČENÁ LITERATURA:

[1] BENANTAR, Messaoud. Access Control Systems : Security, Identity Management and Trust Models. 1st edition. [s.l.] : Springer, 2005. 261 s. ISBN 978-0387004457.

[2] FERRAIOLO, David F., KUHN, D. Richard, CHANDRAMOULI, Ramaswamy. Role-Based Access Control. 2st edition. [s.l.] : Artech House Publishers, 2007. 418 s. ISBN 978-1596931138.

[3] MCCARTY, Bill. SELinux : NSA's Open Source Security Enhanced Linux. 1st edition. [s.l.] : O'Reilly, 2004. 254 s. ISBN 978-0596007164.

Termín zadání:	11.2.2008	***Termín odevzdání:***	28.5.2008

Vedoucí práce: Ing. Tomáš Pelka

prof. Ing. Kamil Vrba, CSc.
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA

POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Miroslav Grepl
Bytem: Nová 1, 78985, Mohelnice
Narozen/a (datum a místo): 21.1.1984, Šternberk

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 244/53, 60200 Brno 2
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Ing. Kamil Vrba, CSc.

(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - ☐ disertační práce
   - ☑ diplomová práce
   - ☐ bakalářská práce

   jiná práce, jejíž druh je specifikován jako .....

(dále jen VŠKP nebo dílo)

Název VŠKP: MAC řízení přístupu
Vedoucí/školitel VŠKP: Ing. Tomáš Pelka
Ústav: Ústav telekomunikací
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

- ☑ tištěné formě - počet exemplářů 1
- ☑ elektronické formě - počet exemplářů 1

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☑ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

ABSTRAKT

Diplomová práce přehledně popisuje problematiku SELinuxu a způsoby vytváření vlastní bezpečnostní politiky se zaměřením na referenční politiku a její mechanizmy. Navrhuje metodiku vytváření konkrétních bezpečnostních pravidel doplněnou o praktické příklady jejich aplikace. Dále popisuje dostupné bezpečnostní pravidla pro zabezpečení http, ftp, ssh služeb, jejich případnou modifikaci a praktické využití. Podle navržené metodiky jsou tyto služby zabezpečeny vlastními pravidly a obě metody zabezpečení vzájemně porovnány a přehledně vyhodnoceny.

KLÍČOVÁ SLOVA

SELinux, bezpečnostní politika, bezpečnostní kontext, bezpečnostní moduly a pravidla, referenční politika, vynucení typu, kontrola založená na roli

ABSTRACT

This master's thesis describes the problems of SELinux, and the methods of creation of a proper security policy with a focus on the SELinux reference policy and its mechanisms. It designs the methodics of formulation of specific security rules, supplemented with the practical example of its application. Furthermore, it describes the available security rules commonly used for http, ftp and ssh services securing, their modification and practical utilization. According to the proposed methodology, these services are protected with their own security rules and both security methods are mutually compared and evaluated.

KEYWORDS

SELinux, security policy, security context, security modules and rules, reference policy, Type Enforcement, Role-Based Access Control

GREPL M. *MAC řízení přístupu* . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 79 s. Vedoucí diplomové práce Ing. Tomáš Pelka.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „MAC řízení přístupu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce za cílené vedení, cenné rady při tvorbě této práce a za čas, který mi věnoval.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V době informačních technologií patří mezi základní požadavky různé formy zabezpečení operačních systémů počítačů, programů, které obsahují mnoho slabin a umožňují velké množství útoků za účelem získání privátních informací uživatelů, narušení integrity dat nebo jejich úplnou ztrátu. Mezi nejrozšířenější softwarové útoky lze zařadit útok založený na přetečení zásobníku (tzv. „buffer overflow“), který může vést v rámci operačního systému Linux k získání práv superuživatele.

Operační systém stojí na pomezí komunikace mezi počítači a aplikacemi, řeší problematiku nebezpečných aplikací a jejich další vliv na chovaní aplikací a uživatelů, může je vhodně omezovat. Jedním z takových způsobů je implementace bezpečnostního systému do operačního systému a specifikace bezpečnostní politiky. Tuto implementaci si lze pak představit jako interní firewall operačního systémů.

Představitelem takového „interního firewallu“ je projekt vytvořený bezpečnostní agenturou National Security Agency (NSA) implementující povinné řízení přístupu v operačním systému – Security Enhanced Linux (SELinux). Základem je flexibilní architektura, která poskytuje mechanismy k vynucování námi nadefinované bezpečnostní politiky. SELinux v současné době poskytuje několik způsobů povinného řízení přístupu, které, zjednodušeně řečeno, vymezují rozsah činnosti aplikacím, čímž zajišťují vysoký stupeň bezpečnosti uvnitř operačního systému.

Problematika SELinuxu je v současné době velice aktuální a oproti minulosti se začíná objevovat více informací potřebných k jeho porozumění. Největší iniciativa je vyvíjena ze strany hlavního vývojáře SELinuxu ve společnosti RedHat, kterým je Dan Walsh. K SELinuxu existují dvě knihy, které popisují všeobecné principy s několika praktickými příklady a jsou implicitně určeny pro dnes již nevyužívanou *example* monolitickou politiku. K aktuální *referenční* politice existuje pouze několik základních informací a úvod k ní je slabě nastíněn v jediné české dokumentaci k SELinuxu. Tato práce by měla, mimo jiné, plnit funkci dokumentace k SELinuxu, respektive k referenční politice, zaměřenou na její praktickou aplikaci.

Cílem této práce je popsat možnosti jedné z implementací Mandatory Access Control (MAC) řízení přístupu, jež nese název SELinux. Práce se je soustředěna na popis dostupných, standardních pravidel pro nejčastěji provozované serverové služby, jakými jsou `ssh,ftp,http` server. Jednotlivé služby budou rovněž zabezpečeny svými vlastními, na míru upravenými pravidly a tyto dvě metody budou vzájemně porovnány a zhodnoceny. Dalším úkolem je návrh metodologie tvorby vlastních bezpečnostních pravidel.

1 KONTROLA PŘÍSTUPU

1.1 Obecný koncept kontroly přístupu

Obecně je kontrola přístupu vnitřní mechanismus ochrany u operačního systému. Tu nejzákladnější formu kontroly přístupu lze pozorovat v Central Processing Unit (CPU) a to při vykonávání instrukcí, které lze vykonat pouze v *supervisor* módu. Dalším příkladem je práce s virtuální pamětí v jádře nebo uživatelské úlohy.[1]

Kontrola přístupu hraje důležitou roli ve stupni administrace. Starý model **„Přístupové matice“** je základem pro současné mechanismy kontroly přístupu. Tento model je tvořen:

- Objekty – zdroje (například hardwarová zařízení, soubory atd.), které potřebují kontrolu přístupu, tj. musí být zpřístupněné v chráněném módu.
- Subjekty – aktivní entity (například uživatelé, procesy), které přistupují k objektům.
- Práva – operace (jako *enable*, *disable*, *read*, *write*, *execute*) na objektech se representují jako přístupová práva.

	Objekty		
	Soubor1	Soubor2	Soubor3
Proces1	čtení, zápis		vykonání, zápis
Proces2	čtení		
Subjekty	Přístupová práva		

Obr. 1.1: Příklad přístupové matice

Model by mohl být zobrazen jako matice $\mathbf{M}$, ve které každý řádek i representuje subjekt i, každý sloupec j representuje objekt j a matice elementů $\mathbf{M}[i, j]$ obsahuje nastavení práv (viz. obr.1.1).[2]

Subjekty zobrazené v matici nemusejí být jen procesy, ale v tomto pojetí bezpečnostní politiky to bývají uživatelé. Matice $\mathbf{M}[i, j]$ by mohla být i prázdná. V operačních systémech by taková matice měla velký počet řádků a sloupců, ale byla by řídká (většina prvků se rovná nule), čímž by vyobrazení globální tabulky bylo zbytečné. Dalším nedostatkem je statičnost této matice, kdy representovaná oprávnění touto maticí jsou v daném okamžiku statické a nelze tak reagovat na potřebnou změnu práv subjektům. [2]

Přístup pomocí matice využívají UNIXové systémy, kde je sada možných oprávnění redukována na *rwx* (čtení, zápis, vykonání) a jsou definovány třídy objektů – vlastník, skupina, ostatní.

1.2 Vývoj bezpečnosti kontroly přístupu v operačních systémech

Dřívější operační systémy měly malou nebo žádnou bezpečnost. Uživatelé mohli přistupovat ke všem souborům nebo prostředkům jen na základě znalosti jména daného prostředku. Naštěstí tato situace netrvala dlouho a nastoupil mechanismus kontroly přístupu. Dominantním typem kontroly přístupu se stal typ známý jako Discretionary Access Control (DAC). Primárním rysem **DAC** jsou individuální uživatelé, často vlastnící dané prostředky, kteří mohou nebo nemohou k oněm prostředkům přistupovat. Jak uvidíme dále, DAC má určité slabé bezpečnostní stránky. K překonání těchto nedostatků byla snaha vyvinout nějaký vhodnější typ kontroly přístupu. Tím se stal **MAC**, tj. povinné řízení přístupu. Jak bylo řečeno, MAC by měl vylepšit a napravit nedostatky DAC. Bohužel vytvoření vhodného a výkonného MAC mechanismu, který je bezpečný a dostatečně flexibilní, není vůbec jednoduchou záležitostí. Způsobů, jak je MAC v Linuxu implementován, je několik a některé z nich budou dále detailně popsány. [3]

Existují tedy dva druhy řízení přístupu:

- DAC – jedná se o primární typ přístupu v UNIXových systémech a jeho hlavní myšlenka spočívá ve vlastnictví procesů a souborů uživatelem, který má nad nimi plnou kontrolu. Jako vlastník může práva k těmto souborům, procesům poskytnout jiným uživatelům. Nejslabším místem tohoto modelu je superuživatel, který má práva k celému systému. Pokud někdo ovládne proces, který běží s právy superuživatele, má pak neomezená práva. Dalším problémem je neefektivní distribuce práv a omezené možnosti nastavování práv. Právě tyto problémy byly hlavní motivací k vývoji jiného typu přístupu, kterým se stal MAC.[3]

- MAC – povinné řízení přístupu doplňuje diskrétní řízení přístupu a je implementován různými způsoby, tj. v žádném případě nenahrazuje DAC (lze ho v rámci dané implementace modifikovat). To jakým způsobem mohou subjekty přistupovat k objektům je dáno sadou pravidel, která se nazývá politika. Ideálním cílem všech politik je poskytnout subjektům co nejmenší možnou množinu oprávnění, postačujících pro jejich správnou funkci.[3]

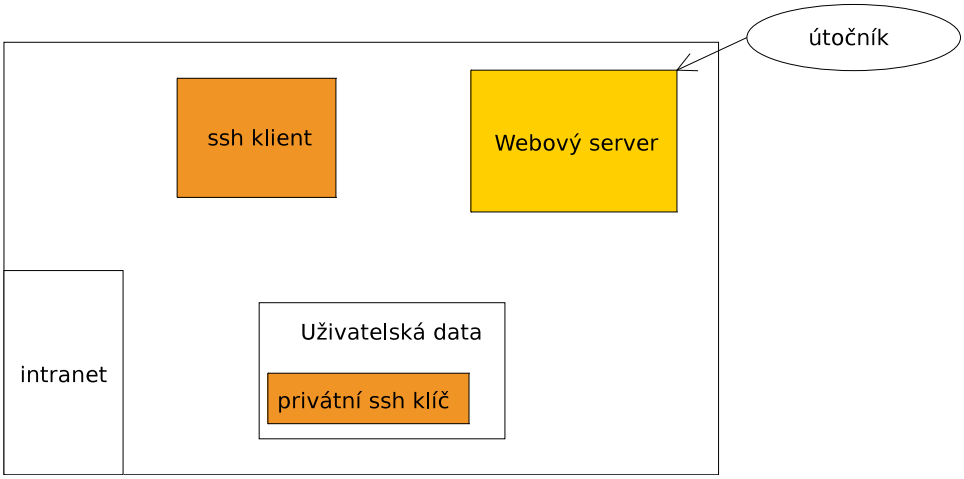
V následující tab. 1.1 jsou pro přehlednost shrnuty vlastnosti DAC a MAC řízení přístupu. Vlastnosti MAC jsou většinou dány zvoleným bezpečnostním modelem a v tomto případě jsou uvedeny obecné vlastnosti MAC.

Tab. 1.1: Vlastnosti DAC a MAC

DAC	**MAC**
politika zakódována v jádře	politiku lze definovat
hrubá granualita (uid,gid)	politiku lze měnit za běhu systému
proces může měnit bezp. atribut	běžné procesy nemohou měnit bezp.atributy
dvě úrovně oprávnění (superuživatel,uživatel)	neexistuje superuživatel (dáno politikou)

2 SELINUX

SELinux je *open-source* produkt, který implementuje povinné řízení přístupu v Linuxu. Byl vytvořen v NSA jako flexibilní architektura pro MAC. Z počátku byl bezpečnostní záplatou na standardní jádro SELinuxu a od řady jader 2.6.x se stal jeho přímou součástí a je dostupný ve většině Linuxových distribucích.[4]



Obr. 2.1: SELinux

Zjednodušeně lze říci, že SELinuxem můžeme definovat jednoznačná pravidla, podle kterých subjekty (uživatelé, programy) mohou přistupovat k objektům (soubory, zařízení). Lze si SELinux představit jako interní firewall, kterému je dána schopnost dělit systém dle pravidel, tj. vymezuje rozsah činnosti aplikacím (viz obr.2.1), čímž zajišťuje vysoký stupeň bezpečnosti uvnitř operačního systému.
SELinux implementuje MAC s využitím následujících mechanismů:

1. Bezpečnostní kontext – bezpečnostní atribut souborů, specifikující soubory v rámci SELinuxu. Základní prvek, podle kterého jsou prováděna rozhodnutí SELinuxu.

2. Role-Based Access Control (RBAC) – subjektům, objektům přiřazuje množinu rolí a změny těchto rolí se kontrolují (a nejlépe jsou povoleny jen v souvislosti s autentizací heslem).

3. Type Enforcement (TE) – role umožňuje procesům (subjektům) vstoupit do určitých domén, v rámci určitých domén je možno manipulovat s určitými typy objektů pouze definovaným způsobem.

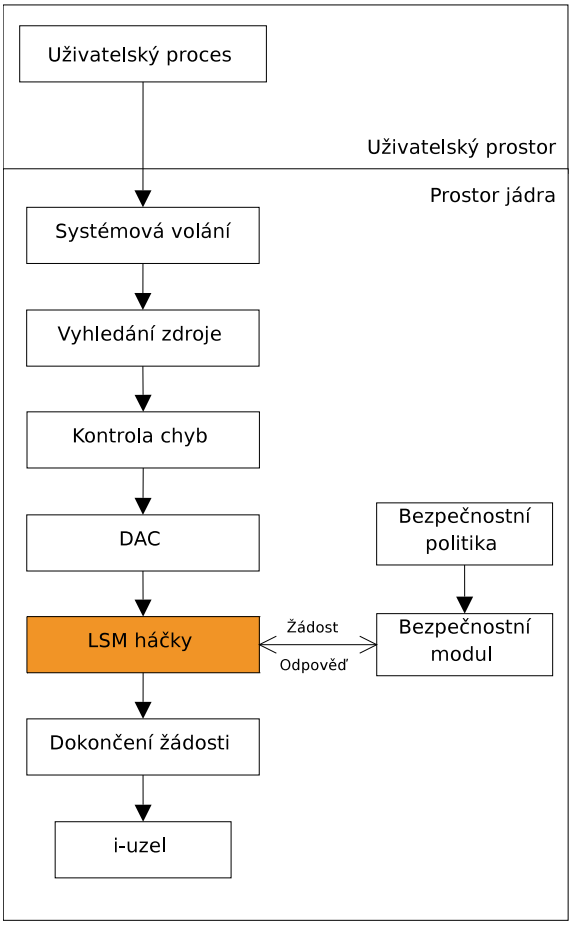
4. Multi Level Security (MLS) – implementace Bell-Lapadula modelu, tj. jsou definovány bezpečnostní úrovně, do kterých jsou objekty podle klasifikace přiřazeny, každý subjekt má své prověření a jsou definována dvě omezení, tj. tento model se zaměřuje na kontrolu toku informací.

2.1 Architektura

Tato kapitola částečně popisuje zmíněnou flexibilní architekturu a jakým způsobem je SELinux implementován v linuxovém jádře prostřednictvím Linux Security Modules (LSM). Zaměřuje se na popis bezpečnostního modelu SELinuxu, tj. jak jsou vynucována pravidla a detailně popisuje bezpečnostní politiku SELinuxu.

2.1.1 Implementace v jádře

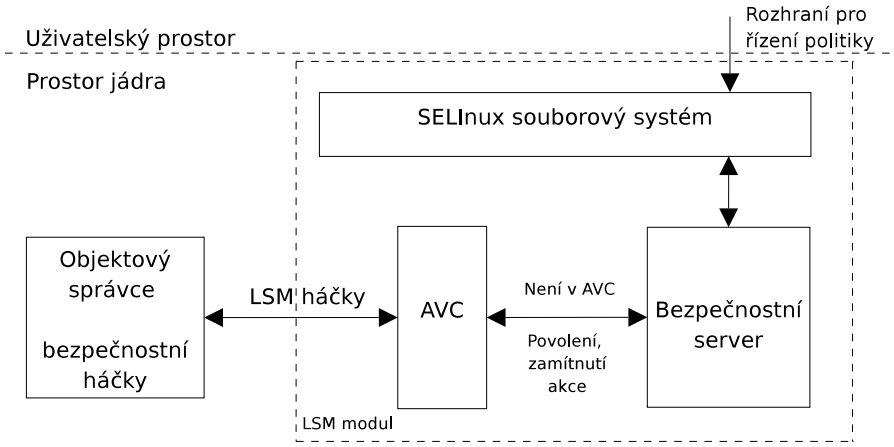
LSM rámec



Obr. 2.2: Architektura LSM *hooks*

Jak bylo naznačeno, SELinux je v jádře implementován jako LSM modul. LSM je zástupcem z řad bezpečnostních rámců, které mají za úkol ulehčit implementaci projektů, jako je SELinux, do jádra Linuxu. LSM záplata poskytuje strukturu (systém) pro podporu řízení kontroly moduly. Samostatně tento rámec neposkytuje žádnou extra bezpečnost. Záplata přidá bezpečnostní oblast (prostor) do datové struktury jádra. Pak se volají háčkové (dále již jen *hooks*) funkce, které jsou vložené v kritických bodech do kódu jádra k řízení těchto bezpečnostních oblastí a vynucování kontroly přístupu. Funkce jsou přidány pro registraci a od registraci bezpečnostních modulů. Tuto strukturu pak mohou využít nahratelné moduly k implementaci různých bezpečnostních politik. Architektura LSM *hooks* je uvedena na obr.2.2.[5]

SELinux jaderná architektura



Obr. 2.3: Architektura v jádře

Základní architektura SELinuxu je zobrazena na obr.2.3. Jedná se o flexibilní architekturu, která implementuje tří základní komponenty: bezpečnostní server, vyrovnávací paměť přístupových vektorů a objektového správce pro vynucení bezpečnostních rozhodnutí. Bezpečnostní server je zodpovědný za rozhodnutí, která provádí na základě bezpečnostní politiky. Ta je nahrávána prostřednictvím rozhraní pro řízení politiky. Objektového správce si lze představit jako susbsystém jádra, v případě LSM se jedná o bezpečnostní *hooks*, které pak vynucují rozhodnutí o možném přístupu či zákazu přístupu k objektům. Třetí komponenta Access Vector Cache (AVC) uchovává rozhodnutí od bezpečnostního serveru pro pozdější kontrolu přístupu, jednoznačně zlepšuje možnost ověření přístupu a je rozhraním mezi objektovým správcem a LSM *hooks*.

2.1.2 Bezpečnostní model

Bezpečnostní model vychází z architektury uvedené na obr.2.3 a zjednodušeně přibližuje činnost SELinuxu Jak bylo již patrné z jaderné architektury SELinuxu, flexibilní architektura rozlišuje mezi bezpečnostními rozhodnutími a vynucovacími funkcemi. Veškerá rozhodnutí jsou vytvářena na základě souborového bezpečnostního kontextu.

V dané architektuře mají všechny subjekty a objekty svůj vlastní bezpečnostní kontext, který specifikuje všechny bezpečnostní atributy subjektů nebo objektů. Jsou užity bezpečnostní identifikátory Security Identifier (SID), které reprezentují jednotlivé bezpečnostní kontexty. Jestliže chce subjekt provést nějakou akci na objektu, jsou pomoci SID nalezeny patřičné bezpečnostní kontexty, které jsou předány serveru vynucení politiky, který vytvoří příslušné rozhodnutí.

Bezpečnostní kontext

Bezpečnostní kontext je základním stavebním prvkem při povinném řízení přístupu v SELinuxu. Bezpečnostní kontext si lze představit jako atribut, který přidělujeme všem souborům (objektům a subjektům) v systému a je uložen v rozšířených atributech souborů. Na základě bezpečnostního kontextu jsou v SELinuxu prováděna bezpečnostní rozhodnutí, jak také zobrazuje bezpečnostní model SELinuxu. Je složen ze tří, resp. ze čtyř částí:

```
SELinux identita : SELinux role : SElinux typ : MLS
system_u:system_r:system_t:s0
```

- Identita – stanovuje, které role může uživatel použít. Je nutné rozlišovat mezi standardní identitou Linuxu reprezentovanou User Identifier (UID) a SELinux identitou. Blíže je tento rozdíl vysvětlen v oddílu 2.2.1.
- Role – určuje, jaká práva má daný subjekt u daného typu.
- Typ – nejdůležitější částí kontextu pro určování přístupu, specifikuje, co všechno s ním lze provádět. Typ procesu nazýváme doménou.
- MLS – Rozšíření bezpečnostního kontextu, udávající citlivost dat, která je v rozmezí `s0 - s15`. Součástí může být i Multi-Category Security (MCS), udávající kategorii objektu (rozsah je v rozmezí `c0 - c255`).

Třídy objektů a povolení

SELinux definuje několik desítek bezpečnostních tříd objektů, které řadíme do hlavních tříd: souborová, meziprocesorová komunikace, síťová a systémová. Příkladem

souborové třídy jsou soubory a adresáře, kterým jsou přiřazeny třídy *file* a *dir*. Akce, které subjekty provádějí na objektech se liší právě tímto typem objektu. Mezi nejčastější operace na souborech, tedy na objektech třídy *file* patří: vytvořit (*create*), číst (*read*), zapsat (*write*) ([4]). Příklady těchto tříd a příslušných operací jsou uvedeny v příloze A.

2.1.3 Bezpečnostní politika

Bezpečnostní politika je sada pravidel, říkající, které role může uživatel použít, které domény mohou vstoupit do jiné domény, které domény mohou používat jaké typy, definice typů atd. (příklady implicitních typů jsou uvedeny v příloze B). V současné době se již používá tzv. **referenční politika**, která nahradila monolitickou NSA SElinux politiku. Veškerá nadefinovaná pravidla v příslušných souborech byla vždy kompilována společně, do jednoho výsledného binárního souboru a každá i malá změna pravidel tak znamenala opětovné překompilovaní všech souborů s pravidly a opětovné zavedení do jádra (LSM modulu).

Referenční politika je politikou **modulární**, tj. pravidla se kompilují do odpovídajících bezpečnostních modulů a ty jsou za běhu systému do politiky nahrávány, popřípadě odstraňovány, bez nutnosti opětovné kompilace celé politiky. Důvodem jejího vzniku byla obtížná administrace předchozí *example* politiky, její komplikovanost, nepřehlednost. Cílem referenční politiky je usnadnit samotnou administraci bezpečnostní politiky, zavést určitý řád při její tvorbě, usnadnit vývoj doplňkových administrativních nástrojů apod. K dosažení takového komplexního systému je využito následujících principů (viz [6]):

- Vrstvení – organizace a snazší pochopení bezpečnostních modulů. Rozlišujeme tyto vrstvy: jaderná, systémová, administrativní, aplikační a vrstva služeb.
- Modulárnost
- Zapouzdření – TE deklarace jsou privátními proměnnými, nikoliv globálními. Příkladem jsou bezpečnostní typy a atributy. Vlastnosti, neboli atributy jednotlivých typů se definují prostřednictvím rozhraní (psaných v makro jazyce *m4*). Zajišťuje tak izolovanost jednotlivých bezpečnostních modulů, umožňuje snazší vrstvení.
- Abstrakce – přispívá ke snížení nutnosti porozumět implementačním detailům politiky. Spočívá ve využití tzv. rozhraní, která jsou volána z příslušného bezpečnostního modulu.

Základní dělení podle typů poskytovaných referenční politikou:

Referenční politika *targeted* implementuje SELinux TE model, tj. kontroluje se výhradně typ. Jde o tzv. nízko úrovňovou politiku a je implicitní po zapnutí SELinuxu (u Fedora Core 5 a výše je SELinux zapnut implicitně). Pravidla (moduly) jsou napsány jen pro několik klíčových aplikací (síťové služby) a zbylé aplikace jsou přiřazeny do tzv. *unconfined_t* , *initrc_t* typu, tj. platí pro ně DAC řízení přístupu.

Referenční politika *strict* k TE modelu přidává RBAC model a kontrolují se tak všechny atributy bezpečnostního kontextu a jde o tzv. vysokoúrovňovou politiku. Implicitní politikou je stav, kdy jsou všechny operace zakázány a povoleny pouze ty, pro která jsou definovány bezpečnostní pravidla.

Referenční politika *MLS* (*MCS*) přidává k vynucování pravidel ještě navíc MLS model. V současné době při instalaci politik je spíše využíván nástavba tohoto modelu – MCS. MCS přiřazuje implicitní citlivost `s0` a objekty dělí do kategorií a pouze subjekty se stejnou kategorií k nim mají přístup. Proces rozhodovaní v jádře je tedy: standardní DAC, TE a RBAC, MCS či MLS.

Rovněž je důležité mít představu o základním modulu bezpečnostní politiky, který je její základní komponentou. Existují dva moduly: `base.pp` a `enableaudit.pp`. Druhý jmenovaný modul má odstraněnu možnost `dontaudit` pravidel (viz oddíl 2.2.2), jinak jsou tyto dva moduly naprosto identické. V daném okamžiku může v politice existovat pouze jeden z těchto modulů. Obsahují definice pro základní typy jako *bin_t*, *etc_t*, *lib_t*, hlavní infrastrukturu modulů. Zahrnuje všechny zdrojové domény a definice jejich typů, jako jsou *kernel_t*, *udev_t*, *crond_t*, *init_t* a nakonec obsahuje uživatelské domény *unconfined_t* (v případě *targeted* politiky), *user_t*, *sysadm_t*, apod.[7]

Bezpečnostní modul

Bezpečnostní modul je tvořen třemi soubory:

1. Privátní politika modulu
   - lokální pravidla a deklarace modulu (`allow` pravidla, volání rozhraní atd.),
   - obsaženy v `.te` souborech.

2. Externí sada pravidel
   - definuji pravidla pro interakci privátních domén/typů s jinými doménami/typy,
   - obsaženy v `.if` souborech.

3. Politika označování souborů
   - definice bezpečnostních kontextů pro dané typy,
   - obsaženy v `.fc` souborech.

Kompilací těchto pravidel získáme výsledný bezpečnostní modul `*.pp`, který lze okamžitě nahrát do aktuální bezpečnostní politiky.

Označování souborů

Jestliže jsme mluvili v souvislosti s bezpečnostním kontextem jako o jednom ze základních kamenů SELinuxu, tak nyní budou zmíněny metody, jakým způsobem v SELinuxu dochází k přidělovaní bezpečnostního kontextu, tj. k označování a podle kterých pravidel je přidělování kontextu řízeno. Platí, že všechny objekty mají přidělen bezpečnostní kontext od jejich vytvoření až po jejich zánik. Jazyk SELinux politiky obsahuje několik možností, například přechodová pravidla pro soubory a domény, které samotné označovaní dělají automaticky a transparentně. V SELinuxu tedy existují čtyři základní možnosti označovaní bezpečnostním kontextem. ([8])

1. Přechodová pravidla specifikující jakým způsobem dochází k označování bezpečnostním kontextem.

2. Implicitní označování zakódované přímo v politice pro mnoho objektových tříd. Například, jestliže proces vytvoří nový soket, tak ten je označen stejným bezpečnostním kontextem jako daný proces. [6]

3. Označování založené na požadavku aplikace. SELinux poskytuje různá Application Programming Interface (API) umožňující programům explicitně vytvářet požadavek na označení, jak pro nové tak i existující objekty. Například pro objekty spojené se soubory podporující označování, využívají tohoto API utility SELinuxu k inicializaci nebo opravě označení.

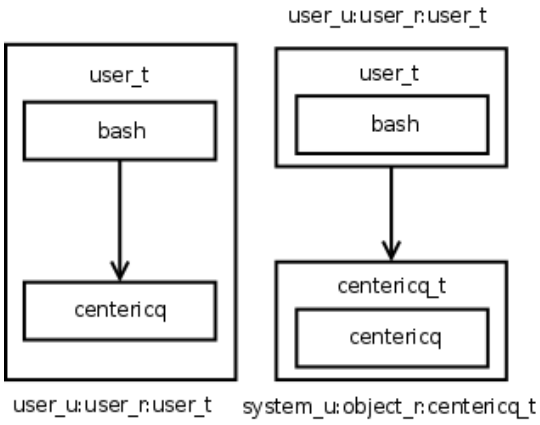
4. Prostřednictvím implicitního SID, který je užíván k označování několika objektů (například k dočasným objektům), nebo jako záchranný kontext, když dojde k nesprávnému označení nebo je chybějící bezpečnostní kontext.

Toto rozdělení není černobíle a k označování souborů se striktně nepoužívají jen tyto základní způsoby. Běžně je nový kontext vypočítán na základě kombinace těchto způsobů. Dalším, již podrobným a obsáhlým dělením metod označování je podle typu objektu, který má být označen. Pro každý tento typ dále existuje několik metod související ze specifikace těchto objektů (viz [6]). Jednotlivé označování dle typů lze dělit na:

- označování spojené se soubory,
- označování síťových a soketových objektů,
- meziprocesní komunikace,

- označování zbývajících objektových tříd,
- implicitní bezpečnostní identifikátor.

Pro názornost je uveden způsob označování procesů běžným způsobem a za použití přechodového pravidla na obr. 2.4. V prvním případě nedoje při spuštění procesu `centericq` k přechodu do jiné domény, tudíž nový proces je označen stejným kontextem. V druhém případě došlo k přechodu do jiné domény prostřednictvím přechodového pravidla a pro nový proces je vypočten bezpečnostní kontext daný pravidlem v politice.

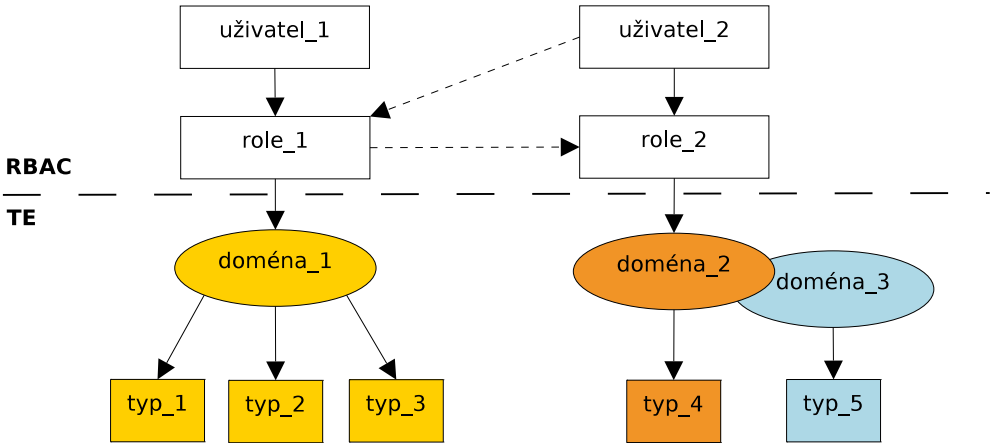


Obr. 2.4: Označování souborů

2.2 Pravidla bezpečnostní politiky

Tato podkapitola popisuje detailněji TE, RBAC mechanismy pro povinné řízení přístupu. Je podrobněji vysvětlena jejich činnost v rámci SELinuxu, jsou uvedeny implicitní deklarace jednotlivých modelů a u komplikovanějších deklarací je využito k jejich zobrazení tzv. railroad diagramu pro lepší názornost a přehlednost.TE přechodová pravidla jsou vysvětlena i na praktickém příkladu, jelikož jejich pochopení je zásadní pro správnou konfiguraci bezpečnostních pravidel.

Pro lepší pochopení souvislostí mezi TE a RBAC při následném popisu pravidel je určen obr.2.5



Obr. 2.5: TE a RBAC

2.2.1 RBAC

RBAC přidává ke standardnímu TE mechanismu další stupeň ochrany, založený na rolích. Každé takové roli jsou z hlediska TE přiřazeny typy, domény do kterých smí role vstoupit a rovněž lze definovat operace, které s nimi role smí provádět. V SELinuxu je definováno několik základních rolí (*user_r*, *staff_r*, *system_r*, *sysadm_r*, *object_r*). Každá role má implicitní politikou stanoveno, k jakým typům smí přistupovat. Nejvíce omezena je uživatelská role *user_r* a přístup ke všem definovaným typům zavedenou politikou má administrátorská role *sysadm_r*.

Při deklaraci nové role uvádíme jméno nové role a příslušné typy, ke kterým má role přístup:

```
role jmeno_role types { sada_typu };
```

Další možnost pravidel je definování přechodů mezi rolemi a to na základě autentizace. Opět tyto přechody jsou pro implicitní role definovány základním modulem. Například není povolen přechod mezi uživatelskou a administrátorskou rolí. K tomuto přechodu je nutné využít role *staff_r* jako mezi článku při přechodu. Při deklaraci těchto přechodů můžeme jedné roli povolit přechod do více rolí:

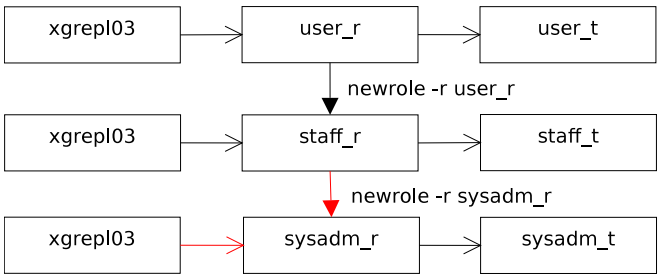
```
allow jmeno_role { sada_roli };
```

Nejčastějším případem RBAC pravidel je deklarace nového, nebo již stávajícího uživatele, kterému přiřazujeme určitou roli. Tento uživatel pak bude oprávněn do zvolené role vstoupit:

```
user SELinux_identita roles { sada_roli };
```

V uvedené deklaraci je nejdůležitější uvědomit si fakt, že pracujeme se SELinuxovými uživateli, nikoliv se standardními uživateli. Příkladem tohoto rozdílu je

superuživatel, který má v Linuxu standardně neomezená práva. V rámci SELinuxu je to obyčejný uživatel a velice záleží na bezpečnostním kontextu, který mu přiřadíme. Jestliže jsme přihlášeni jako běžný uživatel, je nám určitým způsobem přiřazen bezpečnostní kontext (viz. podkapitola 4.1.3) a změna uživatele na superuživatele, například prostřednictvím programu `su`, nezmění SELinux identitu a ani bezpečnostní kontext.



Obr. 2.6: Přechody mezi rolemi

Souvislosti mezi vzorovými deklaracemi, tj. mezi uživateli a rolemi názorně ilustruje obr. 2.6. Mezi jednotlivými rolemi se přepínáme příkazem `newrole`. V posledním kroku je snaha o přepnutí do administrativní role zamítnuta, jelikož uživatel *xgrepl03* nemá oprávnění vstoupit do administrátorské role. Platí pravidlo, že RBAC deklarace definující nové role je vhodné umisťovat do samostatného bezpečnostního modulu, nikoliv například do modulu definující pravidla pro webový prohlížeč. Pokud bychom využívali role deklarované v tomto modulu, pak by jeho odstranění bylo komplikované (role je využita v politice).

2.2.2 TE

TE, nebo-li vynucení typu je základním stavebním prvkem celého SELinuxu. Objevuje se v rámci veškerých dostupných druhů bezpečnostních politik a většina bezpečnostních rozhodnutí je implicitně založena na TE pravidlech. Vynucení typu je prostředkem SELinuxu, jak dosáhnout požadované bezpečnostni celého systému v podobě vymezení činnosti jednotlivým aplikacím (viz obr. 2.1, 2.5). Tj. doménám, typům pomocí pravidel přesně vymezíme přístup k objektům daného typu a vzájemnou interakci mezi doménami.

Subjektům i objektům jsou přiřazeny bezpečnostní typy, které je identifikují a na základě kterých jsou prováděna bezpečnostní rozhodnutí. U subjektu (procesu) je typ označován jako doména. Interakce mezi jednotlivými doménami, typy je určena bezpečnostními pravidly. TE definuje šest základních způsobů deklarace: deklarace typu, aliasu, atributu, přístupových a přechodových pravidel a booleovská. Jednotlivé deklarace pravidel se definují do souborů `*.te`.

Deklarace typu

Typem identifikujeme příslušné objekty a subjekty v systému v rámci TE. Jakým způsobem jsou přiřazovány typy stávajícím objektům a subjektům v systému je definováno bezpečnostní politikou. Deklarace typu v základní podobě:

```
type jmeno_typu;
type httpd_t;
```

Každému typu pak přiřazujeme bezpečnostní atribut prostřednictvím příslušného makra:

```
makro(typ);
files_type(httpd_etc_t);
```

Při deklaraci nových typů je vhodné deklarovat jméno typu co nejvystižněji, běžně ve tvaru `aplikace_t`. Při předeklarování stávajícího typu pro objekt je nutné počítat s problémem, že stávající typ může být využit bezpečnostní politikou. Proto nové typy zpravidla definujeme pro nové aplikace, nebo pro soubory obsažené v domovských adresářích, kde je situace opačná a my nechceme, aby ostatní typy měly k těmto souborům přístup.

Deklarace atributu

Jsou významnou součástí konfigurace pravidel v SELinuxu, ale nutno poznamenat, že běžný uživatel o jejich existenci neví a ani vědět nepotřebuje, což je důsledek zapouzdření a abstrakce v referenční politice. Ovšem pro lepší pochopení pravidel a celkově jejich principu, je vhodné mít o nich alespoň základní představu. V nejjednodušším výkladu atributy seskupují bezpečnostní typy s podobnými vlastnosti. Jak to vypadá prakticky.

Představme si, že v základním zabezpečeném systému je definováno zhruba 800 typů. Ale tento počet bude narůstat s přibývajícími bezpečnostními moduly. Pak mechanismus atributů ulehčuje jazyk bezpečnostní politiky významným způsobem, tj. pomocí jednoho atributu zastřeší i několik desítek podobných typů. Nejběžnějšími atributy jsou *domain* a *file_type*. K těmto zastřešeným typům pak můžeme přistupovat hromadně a nemusíme psát i stovky `allow` pravidel pro každý typ.

Typickým příkladem je administrátorský typ *sysadm_t*, mající přístup ke všem definovaným typům v základní politice. Jestliže přidáme vlastní modul pro webový server s definovaným typem *httpd_conf_t* a nepřiřadíme mu atribut *files_type*, pak administrátorský typ nebude mít k tomuto typu implicitně přístup, pokud nenadefinujeme příslušná pravidla. Celou situaci nejlépe vystihuje tento příklad.

```
avc: denied {getattr} for...scontext:root:sysadm_r:sysadm_t:s0 \
tcontext:system_u:object_r:httpd_conf_t:s0 tclass=file
```

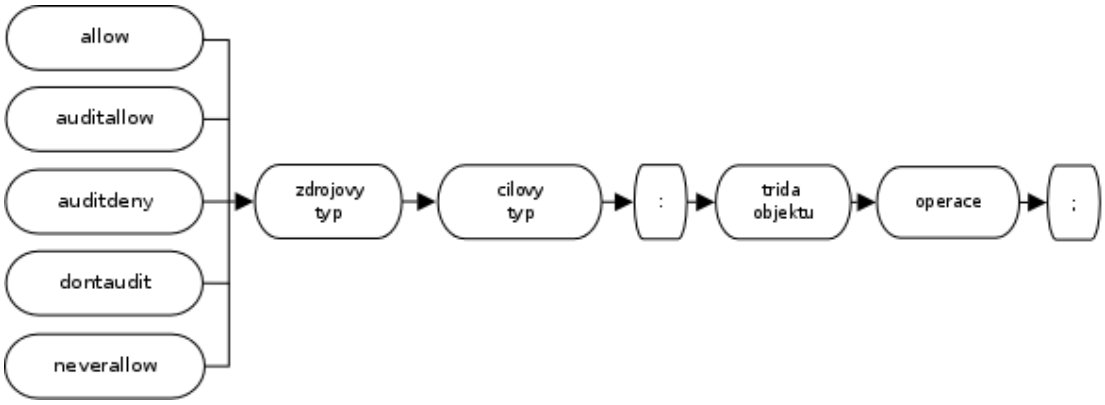
Řešení:

```
type httpd_conf_t; files_type(httpd_conf_t);
```

Přičemž v definici typu *sysadm_t* lze očekávat pravidlo:

```
type httpd_conf_t;
allow sysadm_t file_type:file all_file_perms;
```

Přístupová pravidla



Obr. 2.7: Přístupová pravidla

Přístupová pravidla patří mezi nejvíce využívaný a nejčastější způsob deklarace bezpečnostních pravidel. K bezpečnostním rozhodnutím je využíván příslušný typ a pomocí těchto pravidel jasně definujeme patřičné přístupy (operace) zdrojových typů k objektům určité třídy (např. souborová) s daným typem. Existuje pět způsobů deklarace přístupových pravidel (obr. 2.7). [4]

TE pak vyhodnocuje z typů subjektů, objektů, tříd a druhu operací, zda-li existuje příslušné oprávnění dané přístupovým pravidlem, v opačném případě je akce zamítnuta.
Typy přístupových pravidel:

- `allow` - operace, které jsou povoleny a nejsou zaznamenávány,
- `auditallow` - povolené operace jsou zaznamenávány, nikoliv povoleny,
- `auditdeny` - zakázané operace jsou zaznamenávány, nikoliv zakázány,
- `dontaudit` - nedochází k zaznamenávání nežádoucích operací,
- `neverallow` - zakazuje již povolené operace, je nadřazené povoleným operacím.

V praxi se nejčastěji setkáme s `allow` a `dontaudit` typy přístupových pravidel, zejména prvně jmenovaná možnost tvoří základ bezpečnostních modulů. Chceme-li povolit webovému serveru s typem *httpd_t* číst a zapisovat do souborů s definovaným typem, pravidlo bude mít tento tvar:

```
allow httpd_t httpd_xgrepl03_content_t:file {read write};
```

Přechodová pravidla – soubory

Tento typ přechodu stejně jako pravidla přechodu pro procesy jsou klíčovým prvkem pro správně fungující a bezpečnou konfiguraci. Těmito pravidly definujeme výsledný typ pro nově vytvořené soubory. Obecně lze říci, že označování souborů spočívá na principu dědění typů. Příkladem je vytvoření souboru v domovském adresáři, kdy tento soubor získá stejný typ rodičovského adresáře.
Pravidlo pro deklaraci tohoto přechodu se definuje takto:

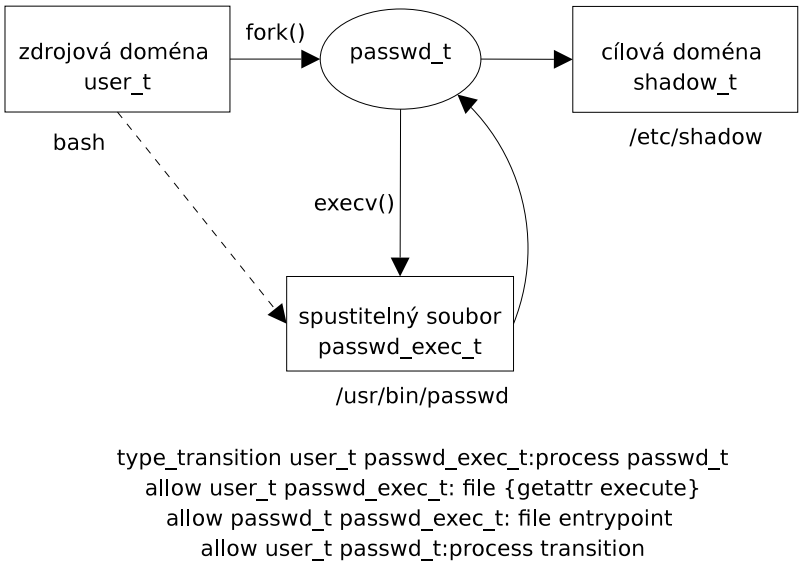
```
type_transition source_type target_type : object_classes name_type;
```

Jako příklad lze uvést vytvoření souboru s informací o Process Identifier (PID) spuštěného webového serveru. Doména *htppd_t* vytvoří příslušný soubor v adresáři `/var/run/` s cílovým kontextem `httpd_var_run_t`. Odpovídající přechodové pravidlo:

```
type_transition httpd_t var_run_t: file httpd_var_run_t;
```

Přechodová pravidla – procesy

Umožňují nám bezpečně změnit, nebo-li bezpečně přejít z jedné domény do jiné domény a vykonat tak požadovanou operaci, již v rámci cílové domény, která má pro tyto operace daná oprávnění. Typickým příkladem je situace, kdy uživatel s typem *user_t* chce změnit své heslo v souboru s cílovým typem *shadow_t*. Mohli bychom použít příslušné přístupové pravidlo, povolující operace mezi těmito typy. Ovšem tím bychom popřeli samotný princip SELinuxu, atak je nutné najít jiný, užívaný způsob. [6].



Obr. 2.8: Doménový přechod

Tímto způsobem jsou právě přechodová pravidla. V tomto případě je nutné vytvořit taková pravidla, umožňující přechod do domény *passwd_t*, která je autorizována k provádění operací se souborem `/etc/shadow`. Celou situaci nejlépe vystihuje obr.2.8, jehož součástí jsou i pravidla umožňující tento přechod.

Pravidla pro bezpečný přechod z jedné domény do další se tedy skládají z následujících kroků:

1. Definice přechodu ze zdrojové domény do cílové. Je to implicitní pravidlo pro přechod, ovšem nic nepovoluje, pouze definuje. Samotný přechod je povolen následujícími body prostřednictvím `allow` pravidel.

2. Umožnění zdrojovému typu spuštění souboru.

3. Povolení cílovému typu přiřadit příslušnému typu povolení. Tj. povolujeme spustitelnému souboru stát se vstupním bodem do cílové domény.

4. Umožnění zdrojovému typu přejít do cílové domény.

Booleans

Představují velice mocný nástroj SELinuxu, jak měnit bezpečnostní politiku za běhu, tj. bez nutnosti nahrávání či odstranění bezpečnostního modulu, ale umožní okamžitou změnu chování bezpečnostního modulu. Předpokládejme, že máme webový server nakonfigurován implicitně s podporou Command Gateway Interface (CGI) skriptů, přičemž bychom tuto službu nechtěli mít aktivovanou vždy. Řešením je

boolean, např. s názvem *httpd_cgi*, které implicitně nastavíme její hodnotu na `1` (zapnuto) a v případě potřeby tuto službu deaktivujeme, nastavíme hodnotu na `0`. Definiční syntaxe je velice jednoduchá:

```
bool bool_name default_value;
```

Zde je velice důležitá implicitní hodnota, kterou *boolean* přiřadíme. V systému *booleans* rozlišujeme dva typy hodnot, hodnoty trvalé a hodnoty aktuální v kontextu aktuálního souborového systému SElinuxu. Implicitní hodnota je hodnotou, na kterou bude příslušná *boolean* vždy nastavena po restartu systému. Lze ji změnit programem `setsebool` včetně správného přepínače:

```
setseboll -P httpd_cgi true;
```

Aktuální hodnota obsahuje hodnotu *momentálního* stavu a hodnotu *čekající*. Aktuální hodnota určuje, zda-li je *booleans* aktivována či nikoliv. Tento stav si můžeme ověřit programem `getsebool`. Druhá hodnota indikuje budoucí stav momentálního stavu a tato hodnota musí být potvrzena. Existuje několik způsobů editace příslušných souborů, například pomocí `setsebool`, který změní současně oba stavy aktuální hodnoty.

```
getsebool httpd_cgi; httpd_cgi --> off;
setsebool httpd_cgi 1;
getsebool httpd_cgi; httpd_cgi --> on;
```

2.3 Rozhraní a makra

Rozhraní (makra) jsou základním prvkem referenční bezpečnostní politiky. Výrazným způsobem zjednodušují a zpřehledňují vytváření bezpečnostních pravidel a jsou další možností pro jejich tvorbu. Makra jsou psány v makro jazyce *m4* a jsou součástí bezpečnostní politiky v souborech `*.spt` a v souborech rozhraní.

```
define('x_file_perms','{getattr execute}')
allow webadm_t httpd_xgrepl03_content_t:file x_file_perms;
```

Příklad definice makra je uveden na prvním řádku výpisu a jeho použití na řádku druhém. Při kompilaci takto vytvořeného bezpečnostního pravidla budou použity příslušné operace v deklaraci přístupového pravidla definované makrem.

Rozhraní

Umožňují přístup k privátně nadefinovaným typům deklarovaných v rámci příslušného bezpečnostního modulu. Tj. pokud budeme potřebovat přistoupit k typům definovaným například v bezpečnostním modulu `ssh` serveru, učiníme tak voláním příslušného rozhraní (makra).

Představme si situaci, kdy máme definovaný bezpečnostní modul pro webový server s určitým typem (například *httpd_t*) a chceme vytvořit novou roli pro administraci webového serveru týkající se například pouhé změny webových stránek a potřebného restartu služby. V takto nově vytvořeném modulu bychom bez využití rozhraní definovali přístupová pravidla takto:

```
allow webadm_t httpd_xgrepl03_content_t:file {read write};
```

Jestliže by někdy v budoucnosti došlo ke změně typu pro webový server, museli bychom změnu provést ve všech ostatních bezpečnostních modulech, kde se tento typ nacházel. Řešením tohoto problému spočívá ve využití rozhraní.

Rozhraní dělíme na *přístupové rozhraní* a *rozhraní šablony*. Definují se v souborech s koncovkou `.if`. Rozhraní šablony bude popsáno v rámci popisu dostupných pravidel pro síťové služby. Definice přístupového rozhraní má tento tvar:

```
define('jméno_makra','definice pravidel')
```

Součástí rozhraní bývá velice často definice makra *gen_require*, které slouží pro deklaraci potřebných typů. Nejběžněji využívaná makra a příklad definice rozhraní jsou uvedeny v příloze C.

2.4 AVC zprávy

Jestliže vykonáme nějakou nepovolenou operaci na objektech, jsou tyto „přestupky“ oznámeny hlášením jádra v podobě zpráv přicházejících z AVC. Porozumění těmto zprávám je velmi důležité pro vytváření vlastních pravidel politiky. Jestliže mluvíme o zprávách, které něco zakazují, tak samozřejmě existují zprávy, kdy jsou operace povoleny. Následuje příklad AVC zprávy:

```
Apr  3 03:26:35 debian kernel: audit(1207185995.494:7):\
avc:  denied  { execute } for  pid=2385 comm="bash" name="apache2"\
dev=sda1 ino=130719  scontext=root:sysadm_r:sysadm_t:s0-s0:c0.c1023\
tcontext=system_u:object_r:initrc_exec_t:s0 tclass=dir
```

Nejdůležitějšími položkami AVC zprávy pro tvorbu pravidel politiky jsou následující:

- `result`
  Uvádí, zda-li operace byla bezpečnostním serverem povolena nebo zakázána, tj. `granted` nebo `denied`.

- `operation`
  Operace, o kterou se subjekt pokoušel, například čtení, zápis nebo vykonání.

- `source`
  Bezpečnostní kontext subjektu, který se snažil provést určitou operaci na daném objektu.

- `target`
  Bezpečnostní kontext cílového objektu.

- `tclass`
  Typ cílového objektu, tj. příslušná bezpečnostní třída.

AVC zprávy jsou generovány do příslušného systémového logu (`/var/log/messages`) nebo přímo do auditovacího logu SELinuxu, k čemuž je nutné mít aktivovaného auditovacího démona `auditd`. Následnou analýzou těchto záznamů vytváříme příslušná bezpečnostní pravidla.

2.5 SELinux a Debian

V předcházející části byly popsány všeobecně platné principy SELinuxu, které jsou společné pro všechny dostupné distribuce. Odlišnosti pak nastávají v podpoře jednotlivých distribucí z hlediska dostupných verzí politik, dostupných nástrojů pro správu, v rozdílu instalace a umístění souborů SELinuxu v rámci systému. Tato práce byla vytvářena prostřednictvím operačního systému Debian GNU/Linux s linuxovým jádrem 2.6.18.5 pouze v základním systému (textovém režimu), tj. bez příslušného X-serveru a bez správce oken.

2.5.1 Konfigurační soubory

Zpravidla mívají ve všech distribucích shodné umístění, což usnadňuje případnou adaptaci pro správcovství SELinuxu na jiné distribuci. Orientace v těchto souborech je velice důležitá, ne-li nezbytná pro korektní činnost SELinuxu dle našich představ. Budou zmíněny opravdu ty nejpodstatnější konfigurační soubory, které lze dělit do tří podskupin podle umístění a účelu:

/etc/selinux/*

- `config`

Je hlavním konfiguračním souborem SELinuxu obsahující ta nejzákladnější nastavení, jako je výběr typu politiky a módu SELinuxu.

```
#Vyber modu SELinuxu
#enforcing
#strict
#disabled
SELINUX=enforcing
#
#Vyber z instalovanych politik
#ref-policy targeted
#ref-policy strict
SELINUX=ref-policy strict
```

Jednotlivé typy bezpečnostních politik již byly vysvětleny. Velmi důležitá je volba SELinux módu, jehož nevhodnou volbou si lze způsobit nechtěné problémy.

enforcing – v tomto módu je vynucována bezpečnostní politika SELinuxu. Co není povoleno, je zakázáno. Příkladem může být snaha uživatele o vypsání obsahu adresáře:

```
#xgrepl03@debian:~$ ls
ls: .: Permision denied
```

permissive – v tomto módu SELinux simuluje činnost *enforce* módu a operace, které by byly zakázány jsou auditovány. Tento mód se doporučuje mít aktivovaný při vytváření bezpečnostních pravidel.

- `refpolicy-(politika)/policy/policy.20`
  Zkompilovaný soubor bezpečnostní politiky.

- `refpolicy-(politika)/context/*`
  Obsahuje implicitní kontexty pro přihlášení uživatelů, pro `ssh` spojení, pro `cron`. Dále záchranný kontext, kontexty pro `initrc` skripty, vyměnitelná zařízení, příchozí pakety.

- `refpolicy-(politika)/seusers`
  Mapování Linuxových uživatelů na SELinux uživatele.

- `refpolicy-(politika)/setrans.conf`
  Překlady pro MLS/MCS úrovně.

- `refpolicy-(politika)/modules/*`
  Jsou zde soubory a moduly, ze kterých je sestavena aktuální politika a ze kterých byla sestavena předchozí politika.

/selinux/*

- `booleans/*`
  Adresář s definovanými *booleans* v rámci aktuální bezpečnostní politiky.

/usr/share/selinux/politika/*

- `*.pp`
  Jsou zde dostupné bezpečnostní moduly příslušné politiky ve zkompilované podobě a připraveny k použití.

- `include/*`
  Obsahuje definovaná makra, soubory rozhraní a `Makefile`, který lze využít ke kompilaci modulu.

- `include/build.conf`
  Konfigurační soubor pro nastavení voleb při kompilaci politiky. Například volby pro nastavení typu politiky (striktní, MLS, MCS), jméno politiky, verze, druh (modulární, monolitická).

2.5.2 Nástroje pro správu

Nástrojů a příkazů pro správu SELinuxu existuje celá řada a ulehčují nám výrazným způsobem administraci SELinuxu. Jsou součástí různých balíků, jejichž verze a názvy se odvíjejí od nainstalované distribuce. Výpis všech nainstalovaných nástrojů lze zjistit příkazem:

```
# man -k selinux
```

Ty nejdůležitější nástroje s jejich popisem shrnuje tab. 2.1.

Tab. 2.1: SELinuxové nástroje a příkazy

Nástroje	**Význam**
`semodule`	řízení SELinuxových modulů instalace, odinstalace, zavedení/obnova politiky
`semanage`	nástroj pro řízení politiky nevyžaduje rekompilaci politiky
`get/setenforce`	aktivace/deaktivace *enforce* módu
`get/setsebool`	zjištění a nastavení stavu *boolean*
`restorecon/fixfiles`	přeoznačení souborů, adresářů celého souborového systému
`audit2allow`	analýza a vytvoření přístupových pravidel
`checkmodule/semodule_package`	kompilace bezpečnostních modulů do binární podoby
`policygentool`	automatické vytvoření souborů `*.te`, `*.fc`, `*.if`

2.5.3 Kompilace politiky

Ke kompilaci i instalaci bezpečnostních modulů se využívají kombinace nástrojů uvedených v tab. 2.1 nebo k samotné kompilaci modulu lze využít i předdefinovaného souboru `Makefile` (viz. oddíl 2.5.1).

- `Makefile`

  1. Vytvoření modulu `*.pp` ze souborů `*.te`, `*.fc`, `*.if`.
     ```
     #make -f /usr/share/selinux/refpolicy-strict/include Makefile
     ```
  2. Zavedení modulu a aktualizace politiky.
     ```
     #semodule -i *.pp
     ```
  3. Přeoznačení souborů nebo celého souborového systému.
     ```
     #restorecon -R -v cesta_k_souborum
     #fixfiles relabel
     ```

- Checkmodule

  1. Vytvoření modulu `*.pp` ze souborů `*.te`, `*.fc`, `*.if`.
     ```
     #checkmodule -M -m proftpd.te proftpd.mod
     #semodule_package -o proftpd.pp -m proftpd.mod
     ```
  2. Zavedení modulu a aktualizace politiky.
     ```
     #semodule -i *.pp
     ```
  3. Přeoznačení souborů nebo celého souborového systému.
     ```
     #restorecon -R -v cesta_k_souborum
     #fixfiles relabel
     ```

2.5.4 Instalace

Instalace SELinuxu je poněkud obtížnější a lze nalézt obecný postup u všech distribucí, kdy rozdíl v instalaci je pak dán například užitým balíčkovacím systémem, dostupnými utilitami. Všeobecně je nutné mít aktivovanou volbu `Security Labels` u vybraných souborových systémů, nainstalovat potřebné pomocné programy, SELinux politiku a SELinux knihovny. Pokud není, vytvořit adresář `/selinux/` a do tabulky `fstab` přidat záznam pro připojení tohoto adresáře, do zavaděče přidat záznam `selinux=1`. Mezi poslední kroky patří označení všech souborů jejich bezpečnostním kontextem. Jak bylo řečeno, postup instalace se pro jednotlivé distribuce liší, ale vždy obsahuje nějaký z těchto kroků.

Následuje příklad instalace SELinuxu na distribuci Debian s jádrem verze 2.6.18.

1. Kontrola, je-li jádro zkompilováno s volbou `Security Labels` u souborových systémů.

2. Instalace potřebných balíčků.

   ```
   # apt-get install <jméno balíku>
   ```

3. Označení souborového systému.

   ```
   # fixfiles relabel
   ```

4. Editace konfiguračního souboru zavaděče k zapnutí SELinuxu.

   ```
   #kernel  /vmlinuz-2.6.18-5 root=/dev/sda<x> selinux=1
   audit=1 ro enforcing=1
   ```

5. Restart systému.

6. Instalace různých utilit pro správu SELinuxu.

2.5.5 Podpora souborových systémů

V typických souborových systémech Linuxu je každému souboru přiřazen jedinečný identifikátor – i-uzel. Ten obsahuje důležité informace (metadata): přístupová práva, seznam datových bloků apod. Když jádro odkazuje na tento soubor, jeho i-uzel je načten do paměti. Standardní UNIXová práva se pak kontrolují na základě informace získané z i-uzlu. Jak již bylo řečeno, SELinux využívá bezpečnostního kontextu k vytváření bezpečnostních rozhodnutí. [8]

Pro uložení těchto bezpečnostních kontextů SELinux implementuje tzv. Extended Attributes (EA), označované také jako EAs nebo *xattr*. Rozšiřující atributy jsou

uloženy uvnitř jmenného prostoru (dále jako *namespace*), povolující rozdílné třídy atributů, které mohou být řízeny separátně. Například Access Control List (ACL) jsou uloženy v `system.posix_acl_access` a v `system.posix_acl_default`. SELinux bezpečnostní kontexty jsou uloženy v `security.selinux` *namespace*. EAs mohou být nastaveny i manuálně pomocí `getfattr` a `setfattr`. [8]
Například zjištění bezpečnostního kontextu pro modul `centericq.pp`:

```
# getfattr -n security.selinux ap.pp
# file ap.pp
security.selinux = "root:object_r:sysadm_home_t:s0\000"
```

Pro souborové systémy k podpoře bezpečnostních kontextů je tedy nutná podpora rozšiřujících atributů a *handlerů* do EA bezpečnostního jmenného prostoru. V současné době je tato podpora dostupná u těchto souborových systémů: ext3, ext2, XFS, ReiserFS, JFS. Kdy a jakým způsobem dochází k inicializaci souborových systémů v rámci SELinuxu je uvedeno v oddílu 4.1.1

/selinux

Tento souborový systém je tzv. pseudo-souborový systém, stejně jako `/proc/` nebo `/sys/`. Zapisují se do něj změny příznaků bezpečnostní politiky, jako aktivace, deaktivace *boolean*, přepnutí módu politiky mezi *permissive* a *enforcing*, pro zjištění stavu různých částí. Je uveden příklad vybrané části tabulky `fstab` s tímto souborovým systémem.

```
$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults          0       0
/dev/sda1       /               ext3    defaults,errors=remount-ro 0   1
none            none            selinuxfs    rw           0       0
```

3 METODIKA NÁVRHU VLASTNÍ POLITIKY

Cílem této kapitoly je co nejlépe popsat tvorbu vlastních bezpečnostních modulů (politiky) prostřednictvím všech popisovaných možností a zároveň vytvořit co nejefektivnější metodiku pro zabezpečení aplikací. Samotná metodika bude šablonou pro vytváření pravidel a snaží se komplexnost problému při vytváření pravidel dělit na menší části a usnadnit tak vytváření bezpečnostních pravidel, resp. modulů. Metodika je pouze doporučením, jak postupovat při zabezpečení aplikací, nikoliv příručkou obsahující podrobná vysvětlení všech použitých mechanismů. Z toho důvodu jsou nutné některé požadavky pro úspěšné zvládnutí této metodiky:

- Pochopení základních principů SELinuxu.
- Práce se SELinuxovými nástroji pro tvorbu, řízení politiky.
- Znalost zabezpečovacích pravidel, zejména přístupová pravidla.
- Kompilace politiky.
- Informace o aplikaci.

Tab. 3.1: Metody vytváření bezpečnostních modulů

Metodika 1	**Metodika 2**
1. Zapnutí *permissive* módu	1. Zapnutí *permissive* módu
2. Vymazání auditovacích záznamů `cat /dev/null >> /var/log/*`	2. Vytvoření souborů modulu `policygentool modul aplikace`
3. Spuštění aplikace	3.Vymazání auditovacích záznamů, spuštění aplikace
4. Užití nástroje `audit2allow`	4. Užití nástroje `audit2allow`
`cat /var/log/* | audit2allow -M soubor`	`audit2allow -R < /var/log/* > soubor.te`
5. Instalace modulu	5. Kompilace a instalace modulu
6. Přeoznačení souborů	6. Přeoznačení souborů

Existuje několik postupů, jak dospět k výslednému bezpečnostnímu modulu plnící požadovanou funkci, které se odvíjejí od znalosti SELinuxu. Ten nejjednodušší spočívá v manipulaci s nástrojem `audit2allow`. Jestliže tedy tímto nástrojem vytvoříme určitá pravidla, z nich modul, který zkompilujeme a zavedeme do politiky, máme tak snadnou metodu, jak rychle a téměř bez znalosti SELinuxu vytvářet bezpečnostní politiku.

Další metodou, již vyžadující částečnou znalost SELinuxu je kombinace nástrojů `policygentool` a `audit2allow`. Touto kombinací dostaneme bezpečnostní modul, již podle zásad referenční politiky, tedy alespoň co se týče struktury modulu (tj. z jakých souborů je modul vytvořen) a také více dynamická pravidla pro danou aplikaci. Oba tyto postupy společně s praktickými příklady zobrazuje tab.3.1. Samozřejmostí při vytváření modulů je mít aktivovaný *permissive* mód.

Uvedené postupy patří mezi nejzákladnější způsoby tvorby bezpečnostní politiky a obdobných kombinací existuje více. Pokud bychom se spoléhali pouze na SELinuxové nástroje a žádným způsobem bychom do samotné tvorby modulů nezasahovali, výsledkem by ve většině případů byla špatně nakonfigurovaná bezpečnostní politika, mnohdy plnící zcela jinou funkci, než-li jsme požadovali.

Zřejmě největším problémem je spoléhání se na program `audit2allow` a na pravidla jim vygenerovaná. Z principu jeho činnosti vyplývá, že pouze povoluje dané operace mezi subjekty a objekty, či-li konfiguruje přístupová pravidla. V příloze D je uveden ukázkový postup a příklady pravidel zabezpečení Internet Messenger (IM) klienta `centericq` podle následujícího návrhu metodiky.

3.1 Návrh metodiky

Při návrhu vlastní metodiky tvorby pravidel bylo vycházeno z výše popsaných předpokladů, ale i nedostatků zmíněných postupů. Navržená metodika se snaží co nejlépe kombinovat stávající postupy a klade důraz na pochopení základních principů SELinuxu a rovněž na základní znalosti zabezpečovaných aplikací. Jestliže tyto principy nebyly zcela jasné z úvodního popisu SELinuxu, budou při ukázkovém použití metodiky uvedeny vždy praktické příklady s vysvětlením, které tyto problémy objasní a pomohou uživateli lépe chápat problematiku SELinuxu, hlavně z hlediska jeho praktického využití.

Celkový postup tvorby bezpečnostního modulu je tvořen čtyřmi základními body, z nichž pak každý zapouzdřuje potřebné činnosti odpovídající názvu tohoto bodu.

1. Informace o aplikaci
   (a) Všeobecné informace o činnosti aplikace
2. Vytvoření počátečního modulu
   (a) Vytvoření souborů `*.fc`, `*.te`, `*.if`
   (b) Definování počátečních typů, atributů a pravidel
   (c) Kompilace a instalace modulu

3. Testování a analýza

   (a) Spuštění aplikace

   (b) Analýza auditovacích zpráv

   (c) Vytváření bezpečnostních pravidel

4. Kompletace modulu

   (a) Kompilace a instalace modulu

   (b) Vytvoření dodatečných pravidel, *booleans*, podmínkové pravidla.

   (c) Definice rozhraní

3.2 Informace o aplikaci

Prvním bodem této metodiky je správné porozumění základní funkci zabezpečované aplikace. Usnadní nám následnou konfiguraci, kdy si výrazným způsobem snížíme množství auditovacích zpráv o zamítnutých operacích. Následují ukázkové specifické vlastnosti aplikací.

1. Prvním důležitým obecným znakem Linuxových systémů je fakt, že vše je soubor. Proces je instancí programu, jenž je spuštěn operačním systémem. Existují programy, které v systému poskytují nějaké služby, většinou jsou spouštěny při startu systému a pokud jejich činnost není ukončena, běží na pozadí. Tyto programy jsou známé jako služby nebo démoni. Další pro nás důležité členění je na programy základní, uživatelské a systémové, umístěné v adresářích `/usr/bin/` nebo `/usr/sbin/`. [9]

2. Téměř všechny síťové služby mají své hlavní konfigurační soubory uloženy v adresáři `/etc/jmeno_služby/` a součástí konfigurace bývají rozšiřující soubory konfigurace, většinou uloženy v příslušných domovských adresářích.

3. Samozřejmostí všech síťových démonů je možnost zaznamenávání činnosti služby do souborů v adresáři `/var/log/`.

4. PID procesu. Každý nově spuštěný nebo již běžící proces je identifikovatelný prostřednictvím PID. Tato čísla procesů jsou unikátní a v daný okamžik neexistuje více procesů se stejným PID. Systém pak určitým způsobem tyto identifikátory přiřazuje a stejně tak toto číslo ukládá do souboru. Pro konfiguraci pravidel SELinuxu je pak důležité umístění těchto souborů. Nejčastěji se nachází ve `/var/run/` nebo v adresáři `/home/.jmeno_aplikace/`.

5. Využívání dynamických (sdílených knihoven) zpravidla umístěných v souborech `/var/lib/`.

6. Prostředky pro síťovou a meziprocesní komunikaci.

Je nutné poznamenat, že toto je zjednodušený popis programů v rámci **zkOS!** Linux a v žádném případě nevysvětluje veškeré aspekty činnosti procesů v tomto systému. Slouží pouze jako zjednodušený návod pro identifikaci souborů, adresářů spojených s danou aplikací a ulehčuje porozumění psaní bezpečnostních pravidel. K vyhledávání informací o službách, aplikacích v kontextu umístění souborů odpovídajících hledanému názvu, lze využít program pro vyhledávání `find`. Získáme tak jasný přehled o absolutních cestách k hledaným souborům.

3.3 Vytvoření počátečního modulu

Bezpečnostní modul je tvořen třemi soubory viz.2.1 a v tomto bodě metodiky konfigurujeme pouze soubory `*.fc` a `*.te`. V případě deklarace typů se jedná o počáteční konfiguraci, kde jsou definovány základní typy korespondující s vytvořeným souborovým kontextem a nejzákladnější přístupová pravidla odpovídají naším znalostem o činnosti aplikace. Pro vytvoření základní struktury všech tří souborů lze využít nástroje `policygentool`. Jeho výhoda spočívá zejména ve vytvoření komentářů v definované syntaxi pro Extensible Markup Language (XML).

3.3.1 Vytvoření kontextu

Již z principu SELinuxu, kde je bezpečnostní kontext stěžejním prvkem bezpečnostní politiky, je vytvoření kontextu nejdůležitějším krokem ve vytváření bezpečnostních modulů. Konfigurací souboru `*.fc` aplikaci definujeme její rozsah. Při definici tohoto souborů bychom měli mít na paměti několik základních skutečností.

Nalezení všech součástí aplikace v souborovém systému a označení zvolených součástí. Platí pravidlo, čím více označíme souborů souvisejících přímo s činností aplikace, tím aplikaci dokonaleji vymezujeme její rozsah. Současně bychom neměli přeoznačovat klíčové soubory (například systémové konfigurační soubory v adresáři `/etc/`, které jsou využívány i jinými aplikacemi. Samozřejmě že to možné je, ale museli bychom takto nově vzniklým typům přiřadit správné bezpečnostní atributy a v nejhorším případě dodefinovat pravidla pro již existující bezpečnostní moduly. Dále je velmi důležité správným způsobem označit příslušné soubory v domovských adresářích.

V ukázkovém bezpečnostním modulu pro aplikaci *centericq* byl soubor s bezpečnostními kontexty vytvořen na základě vybraných informací o aplikaci `centericq` zjištěných příkazem `find` a je uveden v první části přílohy D.

```
debian:/# find -iname *centericq
./home/xgrepl03/.centericq
./usr/bin/centericq
```

3.3.2 Vytvoření typu

V tomto bodě se budeme zabývat konfigurací bezpečnostních pravidel pro povinné řízení přístupu prostřednictvím TE a RBAC pravidel. Součástí `*.te` souboru jsou tedy zejména pravidla týkající se TE, jako definice typů, atributů, přechodových a přístupových pravidel nebo *booleans*. Z hlediska psaní pravidel lze využít maker nebo přístupových a přechodových pravidel nebo jejich kombinace.

Jelikož tento soubor je někdy velice obsáhlý a nesrozumitelný, je součástí metodiky navržen způsob, jak pravidla dělit do odpovídajících skupin, které mají společné určité znaky. Celý tento soubor tedy lze dělit na části A a B:

- Šablona - část A
    1. Definice typů a atributů.
    2. Je-li nutná deklarace externích typů.
    3. Umožnění spouštění služby při startu systému.
    4. Definice přechodových pravidel.
- Šablona - část B
    1. Souborová třída.
    2. Systémová třída.
    3. Třída meziprocesní komunikace a signály.
    4. Síťová třída.

Dělení části B šablony vychází z bezpečnostních tříd objektů, tak jak je definuje SELinux. Rozdělením na tyto části získáme čtyři oddělené skupiny pravidel, které mají svá specifika a velmi často se opakují i v jiných bezpečnostních modulech. K ulehčení analýzy auditovacích zpráv byl vytvořen i na základě této šablony skript `sablona.pl`. Činnost tohoto skriptu lze charakterizovat následovně:

- Rozdělení AVC zpráv do příslušných tříd, lze zobrazit jen vybranou třídu.

- Možnost zobrazení překladu těchto zpráv a opět lze zobrazit jen vybranou třídu.
- Zobrazení příslušných přístupových pravidel.
- Kombinace všech těchto bodů.

Zejména možnost separace jednotlivých zpráv a zobrazení příslušného přístupového pravidla pro danou třídu šablony části B výrazným způsobem ulehčí proces vytváření pravidel. Podrobnější informace o použití lze zjistit z nápovědy k tomuto skriptu.

Počáteční pravidla

Jak bylo řečeno v úvodu této kapitoly a jak je součástí navržené metodiky, je nutné nadefinovat alespoň definice typů a přiřazení atributů. Již v této fázi je základní modul připraven ke kompilace a následné instalaci do bezpečnostní politiky. V takovém případě bychom po spuštění získali velké množství AVC zpráv. V tomto bodě je důležité nespolehnout se na program `audit2allow` a to z jednoduchého důvodu.

```
avc:  denied  { read } for  pid=2539 comm="centericq" name="config"
dev=sda1 ino=414952 scontext=xgrepl03:user_r:user_t:s0
tcontext=system_u:object_r:centericq_home_t:s0 tclass=file

allow user_t centericq_exec_t:file { execute execute_no_trans read };
allow user_t centericq_home_t:dir { getattr read search write };
```

Použití `audit2allow` by znamenalo povolování objektu se zdrojovým typem *user_t* operace s nově vytvořenými typy programu `centericq` (jak zobrazuje uvedený výpis). Takovým způsobem bychom aplikaci nevymezili její činnost v rámci systému, jestliže to tak zamýšlíme. Z tohoto důvodu je důležité k základní definici typů a jejich atributů nadefinovat přechodová pravidla mezi doménami, způsobem popsaným v oddílu 2.2.2 v části věnované přechodovým pravidlům. Příklad těchto pravidel je uveden v příloze D.

3.4 Testování a analýza

V této fázi po kompilaci a zavedení modulu do bezpečnostní politiky dochází k vytváření bezpečnostních pravidel pro správnou a požadovanou činnost modulu. Důležité při testování aplikace je mít aktivovaný *permissive* mód a nejlépe vymazaný soubor s auditovacími záznamy. Pak již testujeme samotnou aplikaci, službu, tj. snažíme se o testování všech jejich možných stavů.

Současně jsou auditovány zprávy o zakázaných operacích, jejichž počet se odvíjí od komplikovanosti samotné aplikace, až po počet implicitně nadefinovaných pravidel pro náš modul. Takový soubor je pak velice nečitelný i při malém množství

auditovacích zpráv a vytváření pravidel pouhým čtením tohoto souboru by bylo velice zdlouhavé a náročné. Proto existují programy, které tuto činnost více či méně usnadní. Příkladem může být program `audit2why` nebo již zmiňovaný `audit2allow`.

Výrazné zjednodušení při analýze AVC lze dosáhnout prostřednictvím vytvořeného skriptu `sablona.pl`, který vychází z navržené metodiky, ale lze ho samozřejmě využít i nejen v této souvislosti. Ve spolupráci s `audit2allow` tak získáváme velice efektivní nástroj pro tvorbu bezpečnostních pravidel. Je nutné ale uvažovat o potenciálních problémech v souvislosti s využitím `audit2allow`.

- Povolení operací, které nejsou nutné pro činnost aplikace.
- Povoleni nežádoucích, potenciálně nebezpečných operací.
- Modul nemusí fungovat korektně i po povolení všech auditovacích zpráv.

Z prvních dvou uvedených bodů je zcela jasné, že pouhé spolehnutí se na dostupné nástroje není ani v tomto případě vhodné. Jako příklad lze uvést situaci, kdy se doména snaží prohledávat domovské adresáře superuživatele nebo dokonce do nich zapisovat, aniž by to bylo nezbytně nutné. Proto analýza a tvorba pravidel je kompromisem v použití dostupných nástrojů a využití vlastních znalostí o aplikaci.

Situace, které mohou nastat v rámci posledního případu jsou vždy nežádoucí z hlediska uživatele a přinášejí značné komplikace při vytváření bezpečnostních pravidel. Příklady takových problémů a jejich řešení je uvedeno v oddílu 4.1.3. Příklad aplikace skriptu na auditovací zprávy a příslušná pravidla jsou uvedená v příloze D.2.

3.5 Kompletace a kompilace modulu

V poslední fázi vytváření bezpečnostního modulu definujeme, v případě potřeby, příslušná rozhraní. Zejména u služeb je to velice důležitá část, jelikož předpokládáme například více správců systému, kterým tímto způsobem velice efektivně vymezíme práva v rámci SELinuxu. Zřejmě nejběžnější druh je přístupové rozhraní pro vstup do dané domény.

V případě bezpečnostních pravidel, které předpokládáme, že se budou v obdobné podobě vyskytovat i v jiných bezpečnostních modulech, je možné a výhodné tato pravidla zapouzdřit do externího rozhraní umístěného vhodně v rámci vrstvení rozhraní SELinuxu. Příkladem jsou nová rozhraní pro odesílání a příjem Transport Control Protocol (TCP)/User Datagram Protocol (UDP) datagramů z daného síťového rozhraní (viz příloha C).

Další způsoby definice pravidel modulu lze nalézt například ve využití *booleans*, podmínkových deklarací. Velice výhodné je použít `dontaudit` přístupových pravidel:

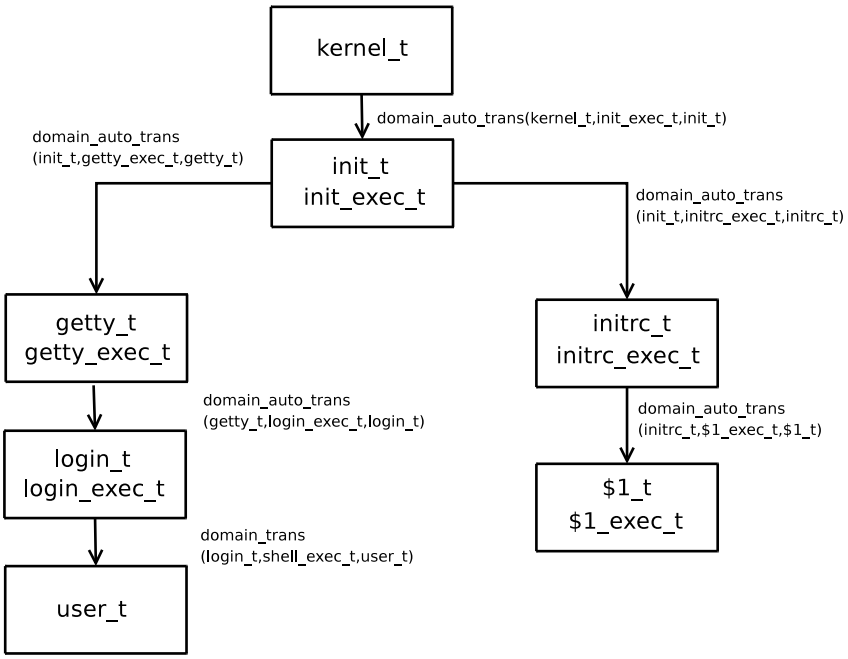
```
dontaudit centericq_t root_home_dir_t:dir getattr;
```

V tomto případě se při každém spuštění aplikace již nebudou auditovat pokusy o získání atributů adresářů superuživatele s daným bezpečnostním typem. Další doplňující možností pro vytvoření modulu patří například podmínkové deklarace, které úzce souvisí s *booleans* a budou popsány v následující části práce. Při správném a kompletním zabezpečení je možné aktivovat *enforce* mód a přesvědčit se o správné činnosti aplikace.

4 ZABEZPEČENÍ SLUŽEB

V této kapitole je uvedena problematika síťových služeb, tj. popis stávajících dostupných pravidel zdrojové politiky SELinuxu a zabezpečení těchto služeb svými vlastními pravidly. Rovněž jsou v úvodu kapitoly vysvětleny a graficky znázorněny důležité pojmy, akce pro správné a regulérní spuštění systému, je stručně vysvětlen způsob označovaní souborových systémů včetně problémů s tím souvisejících a je objasněna problematika implementace řízení přístupu s využitím sítě.

4.1 Zavedení systému



Obr. 4.1: Zavedení systému

Abychom vůbec mohli nějakým způsobem tvořit pravidla, experimentovat, je nutné regulérně spustit systém s aktivním SELinuxem, musí být správně označeny procesy s danou doménou atd. Během načítání jádra se zapnutou podporou SELinuxu má jádro přiřazen doménový typ *kernel_t*. Tento počáteční kontext je využíván před zavedením politiky a všechny procesy spuštěné jádrem mají tento kontext. Následuje spuštění procesu `init`, který načte do jádra politiku a následně ji nahraje. Samozřejmě za předpokladu, že je nalezen parametr povolující zavedení SELinuxu. Tuto možnost lze zakompilovat přímo v jádře nebo stejného efektu dosáhneme editací konfiguračního souboru příslušného zavaděče (viz oddíl 2.5.4). Při úspěšném zavedení politiky dojde k restartu procesu `init` a ten přejde

podle pravidel v zavedené politice do domény *init_t*. K tomtuto přechodu je využito makro *domain_auto_trans(kernel_t, init_exec_t, (target_domain_t))*. Tuto situaci ilustruje obr.4.1. Jsou zde dále znázorněny další doménové přechody, zejména přechody mezi procesy `getty` a `login`.

4.1.1 Inicializace souborových systémů

Během zavádění systému dochází také k inicializaci souborových systémů podle příslušných pravidel. O inicializaci jsme informování zprávou v následujícím tvaru:

```
SELinux: initialized(dev sda3, type ext3), uses xattr
```

Ta nám říká, že souborový systém typu `ext3` užívá k označování souborů rozšířené atributy. Ovšem mnoho souborových systémů rozšířené atributy nepodporuje. Příkladem takovýchto souborových systému jsou `selinuxfs`, `tmpfs`, `devpts` atd. Proto je nutné použít jiné metody pro inicializaci souborových systémů. [8]

- Přechodové SID označování je využíváno pro `devpts`, `tmpfs` a `shmfs` souborové systémy. Soubory na těchto systémech jsou označovány na požádání v jádře, založené na bezpečnostním kontextu aktuální úlohy a bezpečnostního kontextu specifikovaného pro souborové systémy v bezpečnostní politice.
- Úlohové SID označování jednoduše označuje soubory stejným bezpečnostním kontextem jako je kontext aktuální úlohy. Využívá se zejména pro vytváření `pipefs` nebo `sockfs` souborových systémů.
- Označovaní `genfs_context` je využíváno pro souborové systémy nepodporující předchozí možnosti. V bezpečnostní politice jsou soubory označovány bezpečnostními kontexty na základě `soborový_systém/název_cesty`. Nejvíce souborových systémů nepodporující rozšířující atributy využívá této metody.

4.1.2 Proces Init

Po inicializaci je zavedena bezpečnostní politika, jak bylo uvedeno výše a spuštěn proces `init`, jehož úkoly jsou například inicializace operačního systému, operace v adresáři `/tmp/`, spouštění služeb atd. Právě tyto kroky bývají doprovázeny ve *strict* (ale i v *targeted*) politice velkými problémy. V případě implicitně zapnutého *enforce* módu nedoje k úplnému zavedení systému a je nutné vytvořit nový bezpečnostní modul pro korektní start systému. Příklad části takového modulu je uveden v příloze E. Pro názornost a snašší porozumění byly při vytváření tohoto modulu použity, až na výjimky výhradně přístupová pravidla bez použití maker. Nejdůležitější částí

jsou pravidla povolují doménám *init_t* a *initrc_t* vykonávat příslušné operace s adresářem `/tmp/` a ze soubory v něm obsažené. Následuje spouštění příslušných služeb a posledním krokem je spuštění programu `getty`.

Z obr.4.1 jsou tyto dva kroky (body) patrné. První bod vede k úspěšnému přihlášení a získání správného bezpečnostního kontextu. Tento případ je popsán v oddílu 4.1.3. Druhý případ se týká spouštění služeb prostřednictvím `init`. Tento bod je důležitý při konfiguraci bezpečnostních modulů pro dané služby, kdy je nutné definovat přechodová pravidla z domény *initrc_t* do cílové domény služby. Implicitně se k přechodu využívá makra *domain_auto_trans*. V praxi při psaní bezpečnostních pravidel je tento přechod zajištěn makrem *init_daemon_domain*, který zastřešuje i předcházející makro.

4.1.3 Local login

Tato část je v podstatě posledním krokem pro dokončení spouštění systému a jako i předchozí kroky s sebou přináší určité potíže spočívající zejména v nepřiřazení správného bezpečnostního kontextu pro přihlašujícího se uživatele. Tato skutečnost může být zapříčiněna několika možnostmi, které budou nyní diskutovány. V běžném řízení přístupu v Linuxu probíhá přihlašování uživatele po spuštění procesu `login` programem `getty` (viz. také obr. 4.1). `Getty` vypíše na terminálu prompt `login:`. Po zadání uživatelského jména je spuštěn proces `login` a poté za určitých předpokladů (nejsou nastaveny nějaké přídavné restrikce) dojde po zadání hesla k autentizaci a autorizaci uživatele. Po úspěšné autorizaci je spuštěn příslušný příkazový interpret. [9]

V případě SELinuxu je v podstatě situace obdobná, až po autentizaci uživatele a následnou autorizaci, se kterou je spojeno přidělení požadovaného bezpečnostního kontextu. Po úspěšném přihlášení je tedy určitým postupem uživateli přiřazen bezpečnostní kontext.

1. Zadané uživatelské jméno je hledané v `/etc/selinux/*/seusers`.

2. Po nalezení shody je vrácena druhá hodnota odpovídajícího řádku (implicitně po nenalezení shody je linuxovému uživateli přidělen SELinuxový uživatel *user_u*).

3. Nalezení implicitního bezpečnostního kontextu v `/etc/selinux/*/context/ \ users/` nebo v `/etc/selinux/*/context/default_context` a vrátí první možný kontext odpovídající řadku s bezpečnostním kontextem aktuálního procesu
(`system_r:local_login_t`).

4. Uživatel je přihlášen a je mu přiřazen shell s kontextem vypočteným bezpečnostní politikou.

Do celého tohoto přihlašovacího procesu vstupuje velmi důležitý subjekt, kterým je Pluggable Authentication Modules (PAM)., tj. sada knihoven poskytující API pro ověření identity. Popis tohoto mechanismu je nad rámec této práce, ale pro nás je nejdůležitější skutečnost, že hlavním rysem tohoto rozhraní je schopnost dynamicky konfigurovat, jaké aplikace budou používat dané metody autentizace. Funkce jsou rozděleny do čtyř skupin (autentizace uživatele, kontrola účtu, změna hesla, správa relace) a pro nás je nejdůležitější ta poslední. Konfigurace je uložena v Debianu v adresáři `/etc/pam.d/` a příslušnému souboru, odpovídající názvu každé služby, která chce PAM používat.

Stejně jako tento model využívá standardní Linux, tak jej pro specifické případy implementuje i SELinux. Příkladem je využití modulu `pam_selinux_so` pro ověření SELinuxové identity uživatele a přiřazení správného bezpečnostního kontextu. Tato volba je v Debianu implicitně vypnuta a způsobí tak značné problém v podobě nepřiřazení správného bezpečnostního kontextu uživateli. Poznamenejme, že uživateli zůstane kontext procesu `login` (`system_u:system:r_local_login_t`) a nebude mu umožněn přechod do žádné jiné role. Řešením je tedy odkomentovat možnost povolující využití modulu PAM pro SELinux.

```
session required pam_selinux.so multiple
```

Zobrazené nastavení pro správu relace nám tedy umožní přiřazení správného kontextu podle definovaných pravidel úspěšně přihlášenému uživateli. Volba `mulitple` povoluje uživateli vybrat si bezpečnostní kontext, jestliže je tato možnost nakonfigurována. Například superuživateli je implicitně povoleno přihlásit se dvěma bezpečnostními kontexty.

V souvislosti s PAM je možné vysvětlit důvod, proč při změně linuxového uživatele ve striktní politice (i MLS politice) nedojde ke změně SELinuxového uživatele. Program umožňující přihlášení superuživatele `su` nebo programu `sudo` (umožňuje spouštění programů s právy jiných uživatelů) nemají žádný SELinuxový PAM modul. Zapříčinila to skutečnost, že zejména v MLS politice by bylo velmi obtížné určit bezpečnostní úroveň přihlašovaného uživatele a změnit ji.

4.2 Síť v SELinuxu

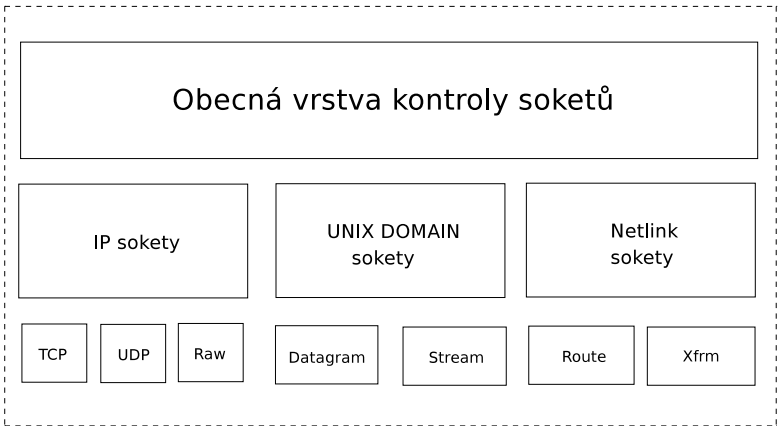
SELinux implementuje prostřednictvím bezpečnostních síťových *hooks* (Netfilter API) rozšířenou ochranu týkající se sítě, tj. SELinux síťovou kontrolu přístupu. *Hooks* jsou umístěni v jádře společně se standardními *hooks* pro kontrolu přístupu, z

čehož vyplývá analogická činnost, daná implementací SELinuxu v jádře (viz obr.2.3). Tato implementace ještě více rozšiřuje schopnost SELinuxu aplikovat povinné řízení přístupu na citlivou část bezpečnosti systémů, serverů, kterou je přístup k síti jako celku. Umožňuje nám tedy přidávat dodatečné restrikce pro síťovou komunikaci, bez ohledu na nastavení síťových firewallů apod. Rovněž pravidla bezpečnostní politiky týkající se sítě tvoří jednou ze stěžejních částí bezpečnostních modulů, zejména modulů pro síťové služby, v tomto případě `http`, `ssh` a `ftp` služby.

Objekty týkající se sítě lze rozdělit do čtyř základních skupin: sokety, síťové zařízení, uzly sítě a porty. Obecně lze říci, že tato třída subjektů je označována prostřednictvím implicitního SID a neexistuje tu mechanismus pro označování založený na požadavku aplikace. Pro jednotlivé skupiny existují specifické způsoby definice pravidel pro označování (viz [6]). Například označení příslušného síťového zařízení nebo portu pomocí SELinuxové utility `semanage` a vylistování všech takto označených rozhraní vypadá následovně:

```
semanage interface -a -t netif_intranet_t eth0
semanage port -a -t ftp_control_port_t 21
semanage interface -l
semanage port -l
```

Samozřejmostí je přítomnost příslušného typu v bezpečnostní politice, který je přiřazen danému síťovému rozhraní, portu prvním uvedeným příkazem. Nejrozšířenějším typem z výše uvedených jsou síťová sokety, které dělíme na několik podtříd, jak znázorňuje obr. 4.2.



Obr. 4.2: Sokety

Významy jednotlivých podtříd jsou dány typem soketu a základní charakteristiky jednotlivých typů uvádí toto rozdělení:

1. UNIX domain sokety – Slouží k zprostředkování přímé UNIXové komunikace, mohou sloužit pro meziprocesní komunikace nebo také pro autentizaci uživatelů. [10]

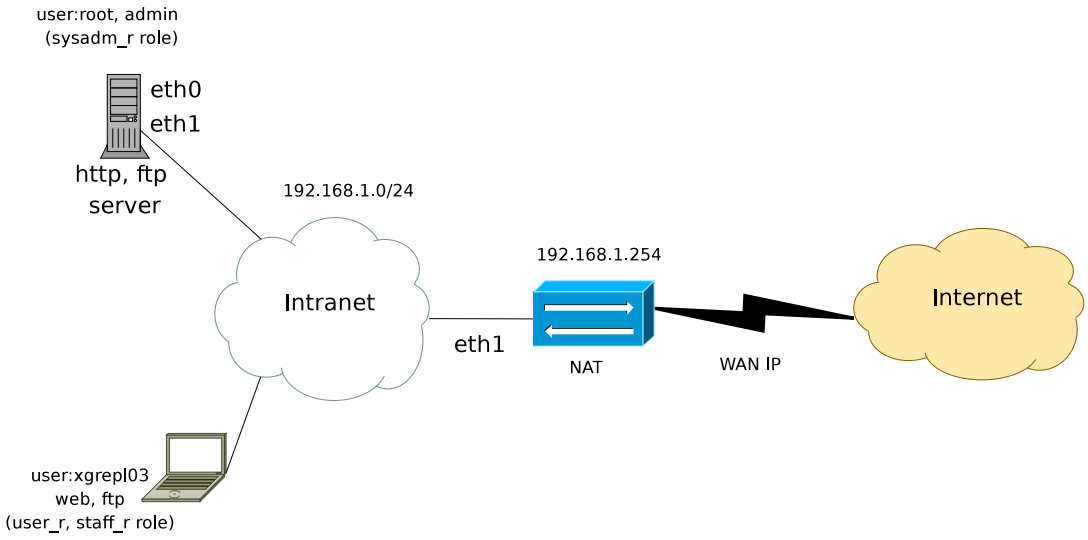
2. Netink sokety – Poskytují komunikaci mezi uživatelem a jádrem, slouží například ke konfiguraci směrovací tabulky nebo Internet Protocol Security (IPSec) mechanismu. Tato komunikace je asynchronní, tj. zprávy se odesílají s určitým kontextem a mohou být přijaté s odlišným. [10]

3. IP sokety – V případě těchto soketů SELinux implementuje různé kontroly pro TCD, UDP a Raw podtřídy soketů. Mezi nejběžnější patří kontrola přijatých a odeslaných paketů přes tyto sokety pro dané síťové rozhraní a uzel. Jsou to například makra *tcp_send*, *tcp_recv* atd. [10]

Správnou konfigurací pravidel týkajících se síťových objektů lze dosáhnout zvýšení bezpečnosti vůči síťovým útokům, ale je nutné uvědomit si fakt, že primární ochranou by měl být korektně nakonfigurovaný firewall.

Pro zabezpečení následujících služeb byl vytvořen na základě výše uvedeného popisu nový modul `rohraní.pp`, který obsahuje definici nového typu pro rozhraní intranetu, v tomto případě `netif_intranet_t`. Rovněž byly vytvořeny makra využívající tohoto typu ke přijmu a odesílání TCP a UDP segmentů.

4.3 Síťové služby



Obr. 4.3: Zabezpečovaná síť

Tato podkapitola si klade za úkol popsat dostupné bezpečnostní politiky zmiňovaných síťových služeb a ty pak zabezpečit svými vlastními pravidly. V rámci

popisu dostupných pravidel bude naznačen možný způsob jejich případné modifikace pro danou konfiguraci dostupnými prostředky, tj. jakým způsobem se prakticky aplikují. Současná pravidla byla vytvořena na míru konfiguraci uvedené na obr.4.3 podle navržené metodiky. V závěrečné kapitole pak tyto postupy budou porovnány a vyhodnoceny jednotlivé rozdíly, výhody či nevýhody každé z těchto metod.

4.3.1 Dostupná politika

Dostupné bezpečnostní moduly obou politik (*targeted*, *strict*) jsou součástí zdrojových kódů politik ve zkompilované podobě a připraveny k použití. V rámci těchto modulů jsou k dispozici příslušné rozhraní, které jsou navrhnuty k případné modifikaci zkompilovaných bezpečnostních modulů dle vlastních požadavků. Pro příslušné soubory rozhraní (`apache.if`, `ssh.if`, `ftp.if`) a nejen pro tyto rozhraní síťových služeb jsou společné mechanismy SELinuxu k definici bezpečnostních pravidel:

- rozhraní šablony,
- rozhraní přístupu,
- *booleans*,
- podmínková deklarace.

Definice (rozsáhlost) těchto prvků se liší modul od modulu a většinou lze najít přímou úměru mezi složitostí služby a rozsáhlostí a komplikovanostípříslušného rozhraní šablony, přístupu a množství využití *booleans* a podmínkových deklarací. Přístupová rozhraní byla již detailně několikrát popsána a jejich studií lze získat částečné informace o vymezení domény v souvislosti s uvedeným typem, ke kterému rozhraní umožňuje přístup. Příkladem je přístupové rozhraní ftp serveru povolující číst konfigurační soubor této služby:

```
interface(‘ftp_read_config’,‘
        gen_require(‘
                type ftpd_etc_t;’)
        files_search_etc($1)
        allow $1 ftpd_etc_t:file { getattr read };’)
```

Template rozhraní

Tvoří nejdůležitější část souborů rozhraní jednotlivých služeb a SELinuxovému administrátorovi umožňují mnoha způsoby upravovat zavedené bezpečnostní moduly služeb podle svých požadavků. Účel těchto rozhraní vysvětlují tyto důvody:

- Dostupné, zkompilované moduly jsou určeny pro určitou konfigurace systému.

  - Jiné umístění konfiguračních souborů.
  - Jiné umístění, jiný typ souborů v domovských adresářích.
  - Vytvoření odvozených typů.

- Nelze při tvorbě některých modulů vědět přesné typy jednotlivých domén pro službu.
  - Vytváření tzv. odvozených domén.

Rozhraní šablony bývají někdy velice rozsáhlá, jelikož musejí obsahovat všechna potřebná pravidla pro vytvoření typu, přechodová a přístupová pravidla atd. Příkladem rozhraní šablony, které za daných okolností nemusí dostačovat požadavkům uživatele, je rozhraní šablony pro `ftp` server.

```
template(‘ftp_per_userdomain_template’,‘
        tunable_policy(‘ftpd_is_daemon’,‘
                userdom_manage_user_home_content_files($1,ftpd_t)
                userdom_manage_user_home_content_symlinks($1,ftpd_t)
                userdom_manage_user_home_content_sockets($1,ftpd_t)
                userdom_manage_user_home_content_pipes($1,ftpd_t)
                userdom_user_home_dir_filetrans_user_home_content\
($1,ftpd_t,{ dir file lnk_file sock_file fifo_file })
        ’)
’)
```

Jak je patrné, toto rozhraní je kombinací booleovské deklarace a využívá maker pro volání příslušných rozhraní. Zjednodušeně si jako výslednou činnost tohoto rozhraní lze představit ve vytvoření dvou typů: *prefix_home_dir_t* a *prefix_home_t*. Prefix je část označení příslušné role, bez identifikátoru role *_r*, tj. pro roli *user_r* je prefix *user*. Doméně *ftpd_t* pak bude umožněné vykonávat se soubory a adresáři těchto typů všechny operace pro tyto objekty, včetně vytváření nových souborů a adresářů.

Nevýhoda tohoto rozhraní je, že jsme limitováni vytvořeným typem pro označení souborů a adresářů, přičemž limitace vychází z požadovaného prefixu. Kdybychom jako prefix užili například *xgrepl03*, což by odpovídalo našim požadavkům pro cílový typ, snaha o zavedení takto zkompilovaného modulu by skončila chybou, jelikož byla využita k vytvoření odvozeného typu neexistující role.

V příloze F je uveden vytvořený modul, který pomocí dostupných pravidel doplňuje zavedené moduly ke správné činnosti služeb dle našich požadavků. Pro jednoduchost jsou v jednom modulu (resp. v souborech `ap2.te` a `ap2.if`) jsou uvedeny příklady úpravy pravidel jak pro `http` server, tak `ftp` server. V praxi by modul měl být separován do dvou modulů. Nejdůležitější částí `ap2.te` souboru je použití vhodného rozhraní šablony. Je nutné zvážit na základě příslušných pravidel v tomto rozhraní, jestli bude odvozený typ, doména plnit požadovanou funkci a zda-li nejsou

zbytečně povoleny operace, které nechceme. Prostřednictvím rozhraní webového serveru byly vytvořeny příslušné typy pro bezpečnostní kontexty webových stránek a CGI skriptů.

```
 apache_content_template(xgrepl03)
# Vytvorene typy timto rozhranim:
# httpd_xgrepl03_content_t
# httpd_xgrepl03_script_t

# Kontext pro soubory webovych stranek
HOME_DIR/www(/.+)? gen_context(xgrepl03:object_r:httpd_xgrepl03_content_t,s0)
```

Rozhraní pro odvozené typy v případě webového serveru nabízí více variant, jakým způsobem typy vytvořit a spoustu možností k ovlivnění bezpečnostních pravidel (booleovská a podmínková deklarace). I přesto může nastat situace, kdy usoudíme, že nadefinovaná pravidla povolují příslušné doméně (například doména pro CGI skripty) příliš mnoho operací a budeme nuceni k vytvoření vlastního modulu pro tyto typy. Takovým případem je rozhraní šablony pro odvozené typy určené k označení adresářů a souborů `ftp` dat. Jelikož tyto typy jsou použity i pro označení ostatních souborů v domovských adresářích, nedoje k požadavané izolaci. Je tedy nutné vytvořit nový typ a nadefinovat příslušná pravidla (viz oddíl 4.3.2). Celý modul upravující pravidla pro `http` a `ftp` server je uveden v příloze F. Pravidla a jednotlivé možnosti modifikace jsou v této příloze detailně popsány.

Podmínkové deklarace a booleans

Součástí rozhraní šablony jsou booleovské deklarace pravidel, resp. využití stávajících *booleans*, pro která jsou definována příslušná pravidla. Rozhraní šablony definují pro odvozené typy pouze nejdůležitější pravidla a pouhé využití těchto rozhraní zpravidla nevede k požadované činnosti. Proto je tedy nutné příslušné rozhraní prozkoumat, zjistit které *booleans* povolují jaké operace a aktivovat je.

```
        tunable_policy(‘httpd_builtin_scripting’,‘
allow httpd_t httpd_$1_content_t:dir r_dir_perms;
allow httpd_t httpd_$1_content_t:file r_file_perms;
allow httpd_t httpd_$1_content_t:lnk_file { getattr read };
        ’)
```

Aktivováním této *boolean* umožníme doméně *httpd_t* číst uživatelské soubory s odvozeným typem, který jsme vytvořili daným rozhraním šablony. Tuto skutečnost je důležité si uvědomit a nespoléhat se na pouhé vytvoření požadovaného typu a označení odpovídajících souborů.

Použití podmínkových deklarací je nejčastěji spjato se systémem *booleans*, které jsou testovaným výrazem. Nejjednodušší podmínkovou deklarací je test na stav *booleans*, jejichž výsledkem je buď `true` nebo `false`.

```
if (podminkovy_vyraz) { pravidla } [  else { pravidla } ]
```

Podmínkové výrazy jsou základní booleovské výrazy používané například v jazyce C (rovnost, nerovnost, logický součin, logický součet a negace).

4.3.2 Vlastní politika

Pro daný konkrétní případ znázorňující obr.4.3 byly vytvořeny vlastní bezpečnostní moduly pro jednotlivé síťové služby, zabezpečeují činnosti těchto služeb v požadované konfiguraci, tj. jsou povoleny opravdu jen nutné operace pro správný běh služeb. K vytvoření těchto modulů byla využita navrhovaná metodika společně se skriptem `sablona.pl`. Z důvodu rozsáhlosti pravidel budou popsány nejzásadnější bezpečnostní pravidla společná všem uvažovaným síťovým službám.

Start služeb

Při definici pravidel je nutné zajistit korektní a úspěšný start služby prostřednictvím procesu `init`. Jakým způsobem dochází k přechodům mezi doménami při zavádění jádra a startu systému bylo vysvětleno na obr. 4.1 a pravidla, která tento přechod zajišťují s jejich popisem v oddílu 2.2.2. Bylo tedy nutné definovat taková pravidla, která umožní doméně *initrc_t* přejít do cílové domény při startu systému. K tomuto účelu u všech služeb bylo zvoleno makro *init_deamon_domain*, které zapouzdřuje několik další maker a patřičná přístupová pravidla.

```
init_daemon_domain(httpd_t,httpd_exec_t)
```

Makro volané tímto způsobem přebírá dva parametry: cílová doména a spustitelný soubor pro přechod do této domény. Nejdůležitější částí tohoto makra jsou makra *domain_auto_trans*, *domain_entry_file* a přístupová pravidla povolující cílové doméně využít popisovače souborů svého rodičovského procesu (zdrojová doména) a následně mu zaslat signál o svém ukončení. Rovněž umožňuje využít meziprocesové komunikace prostřednictvím pojmenované roury, která se využívá ke komunikaci mezi různými procesy (pozn. standardní roury mohou využívat pouze procesy se shodným předchůdcem).

```
interface(‘init_daemon_domain’,‘
        gen_require(‘
                type initrc_t;
                role system_r;’)
        domain_type($1)
        domain_entry_file($1,$2)
  domain_auto_trans(initrc_t,$2,$1)
                allow initrc_t $1:fd use;
                allow $1 initrc_t:fd use;
                allow $1 initrc_t:fifo_file rw_file_perms;
```

```
                allow $1 initrc_t:process sigchld;
                dontaudit initrc_t $1:process {noatsecure siginh rlimitinh};
')
```

SELinux uživatel

V rámci požadavků vyplívajících z obrázku 4.3 byl vytvořen SELinuxový uživatel *xgrepl03*, který je mapován na linuxového uživatele *xgrepl03*. Tento uživatel je oprávněn přecházet z role *user_r* do částečně administrátorské role *staff_r*.

Při vytváření, odstranění či editaci SELinuxového uživatele je doporučeno využívat nástroje `semanage`, který edituje příslušné konfigurační soubory a takto provedené změny se okamžitě projeví v aktuální politice. Přidání SELinuxového uživatele lze provést ve třech krocích:

1. Vytvoření SELinuxového uživatele *xgrepl03* mající přistup do role *user_r* a *staff_r*. Úspěšnost provedení tohoto kroku si lze ověřit rovněž pomocí utility `semanage`.

   ```
   # semanage user -a -R user_r -R staff_r -P user xgrepl03
   # semanage user -l
   ```

2. Mapování SELinuxového uživatele *xgrepl03* na linuxového uživatele *xgrepl03* a následné ověření správnosti tohoto příkazu.

   ```
   # semanage login -a -s xgrepl03 xgrepl03
   # semanage login -l
   ```

3. Úprava bezpečnostního kontextu domovského adresáře uživatele *xgrepl03*.

   ```
   # restorecon -R -v ~xgrepl03
   ```

Přístup k uživatelským datům

Specifikou těchto vlastních pravidel je možnost definice typů a přiřazení je souborům podle požadované konfigurace, nebo-li umístění různých adresářů, souborů služeb a uživatelů. Následně můžeme těmto typům nadefinovat přesně taková pravidla a restrikce, jaké vyžadujeme v rámci daného bezpečnostního modulu.

```
# Deklarace typu
# type xgrepl03_ftpd_home_t;
# Umozneni domene zapisovat, cist do/z souboru a menit obsah adresaru
# allow ftpd_t xgrepl03_ftpd_home_t:file rw_file_perms;
# allow ftpd_t xgrepl03_ftpd_home_t:dir rw_dir_perms;
```

V případě webového serveru rozlišujeme typ pro webové stránky a typ pro CGI skripty umístěných v domovském adresáři uživatele *xgrepl03* a v případě `ssh` serveru je to typ pro soubor s autorizovanými klíči (nejčastěji soubor `authorized_keys`) a pro klíče serveru. Deklarace jsou uvedeny v odpovídajících `*.te` a `*.if` souborech služeb.

Přístup k síti

Pomocí prostředků popsaných v podkapitole 4.2 můžeme velice efektivně omezit vlastními pravidly domény ve vztahu k síťovému provozu, k síťovým zařízením apod. V rámci vlastních pravidel byly definovány restrikce vůči síťovému rozhraní. K tomuto účelu byl vytvořen vlastní nový typ pro síťové rozhraní, kterému nebyl přiřazen žádný atribut, tento typ jsem izolovali od hromadného použití v jiných doménách, nebyl mu přiřazen atribut. Dále byla vytvořena nová makra pro příjem a odesílaní TCP/UDP segmentů.

Pravidla při příjem a odesílaní TCP/UDP segmentů a Internet Control Messages Protocol (ICMP) paketů z/na intranetové rozhraní `eth1` mají tento tvar:

```
corenet_tcp_sendrecv_intranet(httpd_t)
corenet_udp_sendrecv_intranet(httpd_t)
corenet_raw_sendrecv_intranet(httpd_t) # Prijem a odesilani ICMP paketu
```

Modul sshadmin

Stávající pravidla nebo jejich potřebné úpravy a vlastní pravidla zabezpečovali výhradně operace spojené s činností dané služby. Je tak logické, že dostupné bezpečnostní moduly nebo pravidla nezajišťují již žádným způsobem případnou administraci pro dané uživatele. Proto součástí vlastních pravidel byl vytvořen modul, který umožňuje ještě dalšímu uživatele, kromě superuživatele, se částečně podílet na administraci `ftp` a `http` serveru. K definici těchto pravidel bylo využito vlastních rozhraní pro jednotlivé služby, ale i rozhraní poskytovaných nainstalovanou politikou.

Uživateli *xgrepl03* je implicitně přiřazena role *user_t* a je povolen přechod do role *staff_r*. Tato role mu zajišťuje přístup k uživatelským souborům `http` a `ftp` serveru z hlediska jejich modifikace a obě služby může po provedených změnách restartovat. Část zdrojového kódu je uveden v příloze G.

5 SROVNÁNÍ METOD

Kapitola vyhodnocuje obě popsané metody vytváření bezpečnostní politiky, tj. na základě dostupných bezpečnostních pravidel pro nejběžnější serverové služby (`http`, `www`, `ftp`) a na základě vlastních pravidel vytvořených pro tyto služby pro specifickou konfiguraci je vyvozen závěr o jejich náročnosti, vhodnosti použití. Tyto závěry jsou platné v širším slova smyslu, tj. neomezují se výhradně pouze na tyto síťové služby. Jednotlivé klady a zápory každé z metod jsou v závěru této kapitoly shrnuty do přehledné tabulky.

Metody (bezpečnostní pravidla) byly porovnány na základě zvolených kritérií, které nejlépe vystihují jejich rozdíly. Oba druhy bezpečnostních pravidel byly testovány a optimalizovány za použití referenční striktní politiky v obou módech (*permissive*, *strict*) a byla testována jejich základní činnost, tj. úspěšnost spuštění služby při startu systému, její restartování a běžná činnost (poskytovaní daných služeb).

Zvolená kritéria:

1. Funkčnost a analýza.
    - Činnost zkompilovaných bezpečnostních modulů.
    - Modifikace dostupnými pravidly.
2. Rozsah domény
    - Způsoby zjištění rozsahu stávající domény.
    - Aplikace této znalosti.
    - Srovnání domén.
3. Vhodnost metod
    - Znalost SELinuxu, pravidel atd.
    - Obtížnost metod.
    - Časová náročnost (studium stávajících pravidel)

5.1 Kritéria

Navržená kritéria vycházejí z praktických zkušeností při aplikaci obou popisovaných metod pro zabezpečení síťových služeb a měly by být přehledným a zcela jasným ukazatelem pro výběr určité metody nebo modifikace obou těchto metod. K bodu kritéria je vždy uvedena praktická ukázka pro potvrzení zmiňovaných faktů. Jsou uvedeny ty nejobecnější problémy či fakta, která by neměla být závislá na způsobu již

instalované bezpečnostní politiky, kterými jsou například pravidla pro běžný provoz v *enforce* módu.

5.1.1 Funkčnost a analýza

V tomto bodě bude hodnocena zejména vhodnost stávajících pravidel pro danou konfiguraci a vyvození závěru do jaké míry jsou dostačující a jaké je nutné učinit případné modifikace. Tato analýza bude mít za úkol najít co nejpřesnější vymezení problému stávající politiky, pokud nějaké niance nastanou a vytvořit obecný popis nápravy těchto problémů, pokud to bude možné.

Funkčnost

Prvním krokem bylo testování dostupných bezpečnostních modulů na zvolené konfiguraci popsané obr. 4.3. Testovaní bylo provedeno v *enforce* módu. Nebyly přitom uvažovány zakázané operace spojené s administrací systému, pro které nejsou v aktuální politice vytvořené příslušná pravidla. V následujících tab. 5.1 jsou shrnuty výsledné činnosti pro jednotlivé služby a je uveden příklad AVC zprávy informující o nefunkčnosti bezpečnostního modulu pro webový server.

Tab. 5.1: Funkce bezpečnostních modulů

Funkce	**http**	**ftp**	**ssh**
Základní činnost služby	ne	ne	ano
Init	ano	ano	ano
Restart služby	ano	ano	ano
Restrikce vůči síti	ne	ne	ne

Všechny služby byli úspěšně spuštěny při startu systému, bylo je možné restartovat a nebyli jim nastaveny žádné restrikce vůči síti. Problém nastal u `http` serveru a `ftp` serveru v jeho základní činnosti. V případě `ftp` služby nebylo možné se přihlásit na požadovaný účet, nastal problém s autentizací uživatele. V případě webového serveru nebylo možné přistoupit k webovým stránkám a stejně tak nebylo možné spustit CGI skript (viz uvedené výpisy AVC zpráv).

```
avc:  denied  { search } for  pid=2444 comm="apache2" name="www" dev=sda1\
ino=414643 scontext=root:system_r:httpd_t:s0-s0:c0.c1023  \
tcontext=xgrepl03:object_r:httpd_user_content_t:s0 tclass=dir

avc:  denied  { execute } for  pid=2454 comm="apache2" name=  \
"cgi.cgi" dev=sda1 ino=414658 scontext=root:system_r:httpd_t:s0-s0:c0.c1023\
tcontext=xgrepl03:object_r:httpd_user_content_t:s0 tclass=file
```

Analýza

Analýzou takto vzniklých problémů, o kterých jsme informováni AVC zprávami v případě, že aktivujeme *permissive* mód a aplikaci budeme testovat stejně jak v *enforce* módu. Nabízí se několik řešení, které shrnuje tab.5.2

Tab. 5.2: Analýza a modifikace pravidel

Problém	**Modifikace 1**	**Modifikace 2**
Nekorektní činnost služby	Vytvoření odvozeného typu	Vytvoření vlastního typu
Nedostačující pravidla	Využití *booleans*	Definice vlastních pravidel

Obě dvě modifikace byly vysvětleny a názorně předvedeny v oddílu 4.3.1. Součástí práce je také příloha F s ukázkovým modulem `ap2.pp`, který řeší problém základní činnosti webového serveru a částečně `ftp` serveru s využitím navržených modifikací a jejich kombinací.

5.1.2 Rozsah domény

Klíčovým prvkem pro srovnání dostupných a vlastních bezpečnostních modulů je rozsah jejich domén. Vytvořením představy o rozsahu domény získáme mnoho indicií o chovaní bezpečnostního modulu, co vše zabezpečuje a jaké očekává, řekněme implicitní umístění aplikací, jejich konfiguračních souborů atd. Získáním této informace a její vyhodnocení má nesmírný význam pro ulehčení jejich aplikace na požadovanou konfiguraci. Rozsah v tomto slova smyslu zjišťujeme se souboru `*.fc` a u stávajících zkompilovaných modulů je tato informace dostupná po editaci těchto modulů na posledních řádcích `*.pp` souboru.

Tab. 5.3 obsahuje základní srovnání vybraných částí z obou kontextů webového serveru, jelikož jak originální, tak i vytvořený soubor pro označení souborů je rozsáhlý. Vybrané části nejlépe poukazují na rozdílnost obou porovnávaných domén.

Tab. 5.3: Rozsah domény

Rozsah	**Dostupná pravidla**	**Vlastní pravidla**
Domovské adresáře	`HOME\_DIR/((www)|(web)|(public\_html))`	—
Webové stránky	`/var/www(/.*)?`	`HOME\_DIR/www(/.*)?`

Spustitelný soubor	`/usr/sbin/apache(2)?`
	`/usr/sbin/httpd(\backslash.worker)?`
	`/usr/sbin/httpd2\.*`
		`/usr/sbin/apache2`

CGI skripty	`/var/www/cgi-bin(/.*)?`	`HOME\_DIR/www/cgi(/.*)?`

Ze zjištění rozsahu domény tímto způsobem lze vyvodit tyto závěry:

- Požadované umístění souborů v rámci činnosti služby.
- Představa o rozsahu domény.
- Neurčíme rozsah domény v souvislosti s povolenými operacemi.

V souvislosti s prvním bodem je důležité zmínit, že ne každá doména je tak maximalisticky definována, jako ta pro webový server. Někdy nelze přesně odhadnout umístění souborů (například u `ftp` serveru). Není možné určit možná umístění a ty označit daným způsobem, což by mohlo vést k velkým problémům v rámci celé bezpečnostní politiky. Příkladem je `ftp` server a umístění uživatelských souborů, kdy ani dostupné rozhraní šablony není dostačující pro splnění našich požadavků.

Druhým problémem je velmi obtížné určení činnosti aplikace z pohledu definovaných pravidel. Sice máme znalost rozsahu domény, ale již nemáme znalost o pravidlech umožňující či zakazující různé přístupy. Tento problém lze řešit buď testováním služby a částečně lze tato pravidla zjistit z dostupných rozhraní (přístupová rozhraní, *booleans*, rozhraní šablony). Jako příklad uvedeme restrikce vůči síťovému provozu, kdy všechny bezpečnostní moduly mohou přijímat i odesílat provoz ze všech síťových zařízení, síťových uzlů a portů.

5.1.3 Znalost, obtížnost

Jedním z největších problémů při administraci SELinuxu je nutnost pochopení základních principů, které jsou důležité i pro tu nejzákladnější formu administrace, tj. manipulace s instalovanou politikou, nevytváření nových pravidel a korektní činností SELinuxu. Jestliže přihlédneme k faktu, že popsanou problematiku SELinuxu v kapitole 2 bychom mohli považovat za ty nejdůležitější informace pro administraci SELinuxu, je pojem základní znalost velice rozsáhlý.

Budeme-li uvažovat obyčejného uživatele, který chce mít aktivovanou podporu SELinuxu a nemít představu o jeho činnosti, tak tato varianta je možná pouze s *targeted* politikou, nikoliv se striktní politikou. Ulehčení takovému uživateli přináší řada nástrojů, které jsou grafickým rozšířením popisovaných dostupných nástrojů. Tento uživatel se musí plně spolehnout na aplikovanou bezpečnostní politiku v rámci distribuce.

S každým dalším stupněm potřeby přizpůsobit SELinux svým požadavkům, rostou požadavky na znalost SELinuxu a obtížnost těchto kroků. I zde jsou nápomocné grafické nástroje pro tvorbu pravidel. Lze tvrdit, že použití těchto grafických nástrojů souvisí s použitím *targeted* politiky, jelikož na daném systému je potřeba

existence minimálně X-serveru a v současné době provozovat striktní politiku spojenou s uživatelským grafickém systémem je velice náročné a nepoužívá se.

V striktní politice jsou požadavky na znalost SELinuxu velice vysoké a obtížnost její konfigurace je rovněž náročná. Administrátor v tomto případě musí mít co nejlepší znalost bezpečnostní politiky a všech jejich mechanismů a musí být schopen řešit i nepředpokládané situace spojené s činností SELinuxu v rámci operačního systému. Ve spojení s vytvářením vlastních pravidel je nutné přihlédnout také k časové náročnosti, zejména při analýze AVC zpráva a vytváření bezpečnostních modulů. Správné pochopení problematiky a využití SELinuxových nástrojů celý proces tvorby pravidel ulehčí.

Tyto poznatky jsou pro přehlednosti shrnuty v tab. 5.4.

Tab. 5.4: Znalost a obtížnost

Stupeň administrace	**Vhodná politika**	**Znalost**	**Obtížnost**
Základní	targeted	částečná	—
Uživatelský	targeted	základní	střední
Pokročilý	targeted, strict	střední	střední, velká
Administrativní	targeted, strict	pokročilá	velká

5.2 Vyhodnocení

Tato podkapitola shrnuje poznatky a závěry vyvozené z jednotlivých kritérií v předcházející podkapitole a jednoznačně shrnuje klady a zápory jednotlivým metod použití bezpečnostních pravidel.

Tab. 5.5: Srovnání metod

Kritérium	**Dostupná pravidla**	**Vlastní pravidla**
Rozsah domény	maximalistický, dynamický	vymezený
Znalost definovaných operací	minimální	úplná
Funkčnost	nedostačující nutná modifikace	korektní
Znalost	základní – pokročilá	střední – pokročilá
Obtížnost	střední	střední – velká

Z tab. 5.5 lze stanovit následující závěry:

- nekorektní činnost dostupných pravidel je většinou způsobena specifickými požadavky zabezpečovaného systému,

- u stávajících pravidel malá kontrola povolených operací,
- usnadnění aplikace dostupných bezpečnostních modulů dodržením vymezení domény,
- definice vlastních pravidel pro služby v případě požadavku úplné kontroly nad činnosti politiky modulu a v rámci specifických požadavků.

Je patrné, že stávající bezpečnostní politiky pro jednotlivé služby a možnosti modifikace jsou velké, ale ne v každém případě dostačující. Obecně platí, že vhodným způsobem zabezpečování systému, je za pomocí stávajících bezpečnostních pravidel a jejich případnými modifikacemi či dodefinování potřebných pravidel. Zabezpečení svými vlastními pravidly má význam v případě opravdu specifické funkce a vlastním požadavkům na činnost příslušné služby. V obou případech je nutná znalost principů SELinuxu a bezpečnostní politiky, v případě definování vlastních pravidel je větší časová náročnost. Poznamenejme, že ne všechny aplikace jsou zabezpečeny a proto se nikdy nevyhneme definování vlastních bezpečnostních modulů (vlastní domény) jako komplexního celku.

Doporučení pro administraci SELinuxu

Tato doporučení se vztahují pro konfiguraci striktní referenční bezpečnostní politiky SELinuxu pro konfiguraci pravidel zejména na serverech bez využití grafického prostředí, tj. bez využití grafických nástrojů pro správu SELinuxu.

- Znalost přístupových, přechodových pravidel, přehled o dostupných makrech nainstalované bezpečnostní politiky.
- Orientace v konfiguračních souborech SELinuxu a souborech bezpečnostní politiky.
- Využití SELinuxových nástrojů pro správu (`semodule`, `semanage`, `audit2allow`).
- Zajištění korektního startu systému a správné přidělení bezpečnostního kontextu pro přihlašujícího se uživatele viz oddíly podkapitoly 4.1).
- Zvolení vhodné metodiky zabezpečení pravidel dle definice vlastních požadavků a jejich vyhodnocení (využití poznatků z kapitol 3 a 5).
- Konfigurace pravidel a testování vytvořených modulů v *permissive* módu.
- Aktivace *enforce* módu.

6 ZÁVĚR

Problematika SELinuxu je velice komplexní a rozsáhlá. Důležité pro administraci této bezpečnostní implementace je pochopení základních principů pro povinné řízení přístupu v operačním systému, respektive jakými prostředky je dosaženo vynucení bezpečnostní politiky. Tyto základní mechanismy byly názorně vysvětleny a popsány prostřednictvím obrázků. Přehledné definování a vyjasnění těchto pojmů usnadní následnou správu a aplikaci reálného zabezpečení systémů.

Základní funkce SELinuxu je definována jeho bezpečnostním modelem, ze kterého lze stanovit jeden z nejdůležitějších prvků k vynucování politiky – *bezpečnostní kontext*. Kontexty *subjektů* a *objektů* jsou vyhodnoceny bezpečnostním serverem, který obsahuje *bezpečnostní politiku*. V současné době je aktuální politikou *referenční* politika, jejíž základním znakem je modulárnost. Při popisu prostředků k vynucování politik – RBAC, TE byl kladen důraz na přehledné popsání těchto mechanizmů, spojené s praktickými příklady a vysvětlení přínosu referenční politiky. Ten spočívá zejména ve využití zapouzdření a abstrakce prostřednictvím *rozhraní* (*maker*), které umožní před uživatelem skrýt zejména TE deklarace a výsledkem jejich volání je všeobecná činnost, například možnost přístupu ke konfiguračním souborům systému. Řešením kapitoly věnované SELinuxu je tedy příručka jasně vymezující a definující nejzákladnější pojmy s praktickými ukázkami a popisem referenční politiky, ke které existuje pouze několik základních informací od tvůrců této politiky a je zmíněna v české dokumentaci SELinuxu.

Řešením návrhu vlastní metodiky pro tvorbu bezpečnostních pravidel bylo nalezení obecného postupu, který výrazným způsobem zpřehlední a zjednoduší využití dostupných mechanismů k tvorbě vlastní politiky. Koncept vlastní metodiky vychází ze samotných základů SELinuxu, který všechny objekty v rámci systému dělí do jednotlivých bezpečnostních tříd – souborová, meziprocesní, síťová a systémová. Jednotlivá pravidla tak lze definovat v rámci těchto čtyř skupin podle zabezpečované činnosti. Výsledkem aplikace této šablony je zjednodušená analýza auditovacích zpráv skriptem `sablona.pl`, jenž rozděluje tyto zprávy do patřičných skupin, pro které jednotlivě navrhujeme bezpečnostní pravidla.

Při vytvaření bezpečnostní politiky, v tomto případě referenční striktní politiky, máme možnost volit mezi využitím dostupných bezpečnostních pravidel v podobě zkompilovaných modulů, rozhraní, maker a vytvořením vlastních bezpečnostních pravidel. Každý z postupů má své výhody a nevýhody, které jsou dané požadavky na výslednou bezpečnostní politiku. U obou metod je samozřejmostí dobrá znalost principů SELinuxu a jeho nástrojů pro správu. Výběr metody je pak závislý na těchto znalostech, na čase a konkrétních požadavcích.

Výhoda v aplikaci dostupných pravidel spočívá v menší časové náročnosti a ob-

tížnosti, kdy je nutné určitým způsobem modifikovat pravidla, zpravidla pomocí rozhraní, *booleans*. Ovšem mohou nastat případy, kdy jsou potřebné rozsáhlejší modifikace. V případě vlastních pravidel je požadavek větší časové náročnosti a také obtížnosti při jejich tvorbě. Pravidla přesně vymezují činnost služby nebo aplikace pro specifickou konfiguraci.

SELinux je zajímavým a perspektivním řešením pro zabezpečení systémů, ovšem v současné době spíše vhodného pro servery, než-li pro desktopy a to zejména ve spojení se *striktní politikou*. V případě desktopů mluvíme převážně o politice *targeted*. Jistou výhodou desktopů je možnost využití grafických nástrojů pro správu SELinuxu, pokud tedy uvažujeme pouze o základním sytému na serverech. Budoucí vývoj by bylo možné směřovat k vývoji aplikace, založené na navržené metodice, která by usnadnila a korektním způsobem zautomatizovala činnost vytváření vlastní bezpečnostní politiky, nejen pro servery. Jeden z nějvětších problémů požadované automatizace by spočíval ve vytváření přechodových pravidel.

LITERATURA

[1] SINGH, Amit. *Kernelthread.com : A Taste of Computer Security* [online]. Dostupný z WWW: <http://www.kernelthread.com/publications/security/ac.html>.

[2] FERRAIOLO, David F., KUHN, D. Richard, CHANDRAMOULI, Ramaswamy. Role-Based Access Control. 2st edition. [s.l.] : Artech House Publishers, 2007. 418 s. ISBN 978-1596931138.

[3] TURNBULL, James. *Expert Knowledgebase : DAC and MAC safety* [online]. 2006 [cit. 2007-12-10]. Dostupný z WWW: <http://expertanswercenter.techtarget.com/eac/knowledgebaseAnswer/0,295199,sid63_gci1223777,00.html>.

[4] MACCARTY, Bill. SELinux : O'Reilly, 2004. 254 s. ISBN 0-596-00716-7.

[5] WRIGHT, Chris M.: *Linux Security Module Framework*. In Proceedings of the Ottawa Linux Symposium. Ottawa, Ontario. 2002. s. 604-618.

[6] MAYER, Frank, MACMILLAN, Karl, CAPLAN, David. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2006. 456 s. ISBN 978-0-13-196369-6.

[7] WALSH, Dan. *Danwalsh's Journal* [online]. 2006. Dostupný z WWW: <http://danwalsh.livejournal.com/>.

[8] MORRIS, James. *LINUX JOURNAL : Kernel Korner - Filesystem Labeling in SELinux* [online]. 2004. Dostupný z WWW: <http://www.linuxjournal.com/article/7426>.

[9] PELKA, Tomáš. *Návrh, správa a bezpečnost počítačových sítí : Počítačová cvičení*. 2008. 83 s.

[10] MORRIS, James. Architecture of SELinux Network Access Controls. *SELinux Symposium* [online]. 2005. Dostupný z WWW: <http://74.125.39.104/search?q=cache:ouNYZxiL18EJ:selinux-symposium.org/2005/presentations/session2/2-2-morris.pdf+Network+in+SELinuxhl=csct=clnkcd=6gl=cz>.

SEZNAM ZKRATEK

ACL Access Control List

API Application Programming Interface

AVC Access Vector Cache

CGI Command Gateway Interface

CPU Central Processing Unit

DAC Discretionary Access Control

EA Extended Attributes

ICMP Internet Control Messages Protocol

IM Internet Messenger

IPSec Internet Protocol Security

LSM Linux Security Modules

NSA National Security Agency

MAC Mandatory Access Control

MCS Multi-Category Security

MLS Multi Level Security

PAM Pluggable Authentication Modules

PID Process Identifier

RBAC Role-Based Access Control

SELinux Security Enhanced Linux

SID Security Identifier

TCP Transport Control Protocol

TE Type Enforcement

UDP User Datagram Protocol

UID User Identifier

XML Extensible Markup Language

SEZNAM PŘÍLOH

A BEZPEČNOSTNÍ TŘÍDY OBJEKTŮ

Tab. A.1: Souborová třída

Třída	**Význam**	**Operace**
file	Soubor	read (čtení), write (zápis), execute (vykonání), getattr (zjištění atributů), setattr (změna atributů – chmod), append (přidání), link (pevný odkaz)
dir	Adresář	
chr_file	Charakteristické zařízení	
blk_file	Blokové zařízení	
fd	Popisovač souborů	
lnk_file	Symbolický odkaz	

Tab. A.2: Třída procesů

Třída	**Význam**	**Operace**
capability	SELinux *capabilities*	chown (změna vlastníka), dac_override (přepsání DAC rozhodnutí)
process	Proces	signal, sigchld, signull (odesílaní příslušných signalů)
security	Bezpečnostní objekty	check_context, compute_relabel (zapsání kontextu, nastavení přeoznačení v `selinuxfs`)

Tab. A.3: Meziprocesní trída

Třída	**Význam**	**Operace**
msgq	Fronta	read(čtení) write(zápis), create (vytvoření) fronty, semaforu, sdílené paměti
sem	Semafor	
shm	Sdílená paměť	

Tab. A.4: Síťová třída

Třída	**Význam**	**Operace**
socket	soket	accept(přijmutí spojení), bind(pojmenování soketu), connect(zahájení spojení)
Rawip	Raw IP soket	
TCP socket	TCP soket	
UDP socket	UDP soket	

B BEZPEČNOSTNÍ TYPY

V následující tabulce jsou uvedeny příklady implicitních bezpečnostních typů SELinuxu.

Tab. B.1: Příklad bezpečnostních typů

Typ	**Bezpečnostní typ pro:**
device_t	zařízení
devtty_t	zařízení `tty`
null_device_t	`/dev/null`
bin_t	binární spustitelné soubory
etc_t	stálé soubory v adresáři `/etc/`
etc_runtime_t	měnící se soubory v adresáři `/etc/`
fs_t	implicitní typ pro souborové systémy
lib_t	moduly, knihovny a podobné soubory v `/lib/`
shlib_t	sdílené knihovny v `/lib/`
var_runt_t	soubory ve `/var/run/`
user_home_t user_home_dir_t	všeobecné domovské soubory a adresáře
home_root_t sysadm_home_t	adresáře superuživatele
netif_t	síťové zařízení
netif_lo_t	síťové zařízení `lo`
node_lo_t	IP adresu pro `loopback` zařízení
node_t	implicitní typ síťového uzlu
port_t	port TCP/IP
default_context_t	soubor s implicitními kontexty `default_context`
policy_config_t	soubory v `/etc/security/selinux/*`

C MAKRA A ROZHRANÍ

Tab. C.1: Makra s oprávněními pro soubory

Název makra	**Příslušná povoleni**
x_file_perms	vykonání souboru {getattr execute}
r_file_perms	čtení a vykonání souboru {read getattr lock execute ioctl}
ra_file_perms	čtení a přidávání do souboru {ioctl read getattr lock append}
create_lnk_perms	vytváření odkazu na soubor {create read getattr setattr link unlink rename}
create_file_perms	vytvoření a manipulace se souborem {create ioctl read getattr lock write setattr append link unlink rename}

Tab. C.2: Makra s oprávněními pro sokety

Název makra	**Příslušná povoleni**
rw_socket_perms	užívání soketu {ioctl read getattr write setattr append bind connect getopt setopt shutdown}
create_socket_perms	vytvoření a užívání soketu {create rw_socket_perms}
connected_socket_perms	vytvoření a užívání soketu {create ioctl read getattr write setattr append bind getopt setopt shutdown }
connected_stream_socket_perms	vytvoření a užívání soketu {connected_socket_perms listen accept}

Tab. C.3: Makra pro přechod mezi doménami

Název makra	**Příslušná povoleni**
domain_trans(doména, typ, nová doména)	přechod přes spustitelný soubor s typem do nové domény
domain_auto_trans(doména, typ, nová doména)	automatický přechod mezi doménami

Příklady rozhraní a definice příslušných maker, které obsahují. Pro přehlednost jsou makra uvedena bez XML syntaxe.

Rozhraní domain.if

Rozhraní (makra) pro doménový přechod, automatický doménový přechod a pro označení souboru jako vstupního bodu do domény.

```
template(‘domain_trans’,‘
allow $1 $2:file { getattr read execute };
allow $1 $3:process transition;
dontaudit $1 $3:process { noatsecure siginh rlimitinh };
’)

template(‘domain_auto_trans’,‘
domain_trans($1,$2,$3)
type_transition $1 $2:process $3;
’)

interface(‘domain_entry_file’,‘
gen_require(‘
attribute entry_type;
’)
allow $1 $2:file entrypoint;
allow $1 $2:file rx_file_perms;

typeattribute $2 entry_type;

corecmd_executable_file($2)
’)
```

Rozhraní corenetwork.if

Rozhraní (makra) pro odesílání a příjem TCP/UDP segmentů přes intranetové rozhraní `eth1`, kterému byl přiřazen nově vytvořený typ.

```
interface(‘corenet_udp_sendrecv_intranet’,‘
gen_require(‘
attribute netif_type;
type netif_intranet_t;
’)

allow $1 netif_intranet_t:netif { udp_send udp_recv };
’)

interface(‘corenet_tcp_sendrecv_intranet’,‘
gen_require(‘
attribute netif_type;
type netif_intranet_t;
’)

allow $1 netif_intranet_t:netif { tcp_send tcp_recv };
’)
```

D ZABEZPEČENÍ CENTERICQ

D.1 První část přílohy Zabezpečení centericq

1. Vytvoření počátečního modulu - `.fc` a `.te` soubory:

```
# Oznaceni spustitelneho souboru centericq
/usr/bin/centericq --gen_context(system_u:object_r:centericq_exec_t,s0)

# Oznaceni .centericq v domovskem adresari vlastnim kontextem -> izolace
/home/xgrepl03/\.centericq -d gen_context(xgrepl03:object_r:centericq_home_t)
/home/xgrepl03/\.centericq/(.+)? gen_context(xgrepl03:object_r:centericq_home_t)
```

```
policy_module(centericq,1.0.0)

########################################

# Ukazkovy bezpecnostni modul navrzeny podle vytvorene
# metodiky a skriptu sablona.pl.
# Jsou uvedeny priklady pravidel s makry a bez maker
# pro allow pravidla a prechodova pravidla

#### SABLONA - CAST A ####

# Deklarace typu a atributu pomoci maker #
type centericq_t;
domain_type(centericq_t)

type centericq_exec_t;
files_type(centericq_exec_t)

type centericq_home_t;

# Init a prechodova pravidla #

# Allow pravidla bez makra povolujci domenovy prechod
# z domeny user_t do domeny centericq_t.

allow user_t centericq_exec_t:file { read getattr execute };
allow centericq_t centericq_exec_t:file entrypoint;
allow user_t centericq_t:process transition;
type_transition user_t centericq_exec_t:process centericq_t;
dontaudit user_t centericq_t :process {noatsecure siginh rlimitinh};
role user_r types centericq_t;

# Odpovidajici makra

#domain_auto_trans(user_t, centericq_exec_t, centericq_t)
#domain_entry_file(centericq_t, centericq_exec_t)
#role user_r types centericq_t;


# Deklarace externich #
# Pokud nevyuzijeme maker v pripade deklarace pristupovych
# pravidel bez maker
```

```
require{
type user_home_t;
type user_home_dir_t;
type user_t;
};

#### SABLONA - CAST B ####

## SOUBOROVA TRIDA ##

#a) domovske adresare bez maker

# Umozni pristup domene centericq_t do dom. adresaru s danym typem

allow centericq_t centericq_home_t:file {getattr read write append \
create unlink};

allow centericq_t centericq_home_t:dir {getattr search read write \
add_name remove_name};

# Nutna cast pro umozneni domene prohledavat cely domovsky
# adresar s obecnym typem

allow centericq_t user_home_t:dir {getattr search};
allow centericq_t user_home_dir_t:dir {getattr search};
allow centericq_t user_t:dir search;

#b) domovske adresare s makry

# Nutna cast pro umozneni domene prohledavat cely domovsky
# adresar s obecnym typem

#userdom_search_generic_user_home_dirs(centericq_t)
#userdom_search_user_home_t_dirs(centericq_t)
#files_list_usr(centericq_t)

# Neauditovani pokusu pro pristup do adresare superuzivatele

#userdom_dontaudit_root_home_dirs(centericq_t)

#### SYSTEMOVA TRIDA ####

Deklarovana po testovani aplikace a analyze auditovacich zprav.

#### IPC TRIDA ####

Deklarovana po testovani aplikace a analyze auditovacich zprav.

#### SITOVA TRIDA ####

Deklarovana po testovani aplikace a analyze auditovacich zprav.
```

D.2 Druhá část přílohy Zabezpečení centericq

Příklad AVC zpráv, jejich rozdělení do tříd pomocí skriptu `sablona.pl` a vytvoření odpovídajícího pravidla.

```
# ./sablona.pl audit_log log_out
# cat log_out

########## PATRI DO SOUBOROVE TRIDY #######

avc:  denied  { read } for  pid=2539 comm="centericq" name="resolv.conf
dev=sda1 ino=130756  scontext=xgrepl03:user_r:user_t:s0
tcontext=system_u:object_r:net_conf_t:s0 tclass=file

########## PATRI DO PROCESOROVE TRIDY #######

########## PATRI DO IPC TRIDY #######

########## PATRI DO SITOVE TRIDY #######

avc:  denied  { udp_send } for  pid=3525 comm="centericq" saddr=172.16.40.129 \
src=32769 daddr=172.16.40.2 dest=53 netif=eth0 scontext=xgrepl03:user_r:user_t:s0 \
tcontext=system_u:object_r:net_conf_t:s0 tclass=netif

avc:  denied  { tcp_recv } for  saddr=64.12.161.153 src=5190 daddr=172.16.40.129 \
dest=60055 netif=eth0 scontext=root:sysadm_r:centericq_t:s0-s0:c0.c1023 \
tcontext=system_u:object_r:netif_intranet_t:s0 tclass=netif
```

Vytvoření pravidel z těchto auditovacích zpráv, jelikož všechny tři operace jsou nutné pro korektní činnost aplikace. Opět jsou uvedena přístupová pravidla a příslušné ekvivalenty maker.

```
## SOUBOROVA TRIDA ##

# Domena centericq_t muze zjistit informace o jmenny serverech

allow centericq_t net_conf_t:file {getattr read };
#sysnet_read_config(centericq_t)

#### SITOVA TRIDA ####

# Povoleni prijmu a odesilani TCP/UDP segmentu z/na rozhrani eth1

allow centericq_t netif_intranet_t:netif {tcp_recv udp_send};
#corenet_udp_sendrecv_intranet(centericq_t)
#corenet_tcp_sendrecv_intranet(centericq_t)
```

Ukázka rozhrani `centericq.if` pro jednoduchost bez XML syntaxe.

```
# Rozhrani umoznuje pristup (domenovy prechod) do domeny centericq_t
interface(‘centericq_domtrans’,‘
gen_require(‘
type centericq_t, centericq_exec_t;
’)
domain_auto_trans($1,centericq_exec_t,centericq_t)
allow $1 centericq_t:fd use;
allow centericq_t $1:fd use;
allow centericq_t $1:fifo_file rw_file_perms;
allow centericq_t $1:process sigchld;
’)
```

E KOREKTNÍ START SYSTÉMU

```
 policy_module(boot,1.0.0)

# Pro vznik modulu bylo pouzito nastroje audit2allow
# bez upravy pravidel

# Externi atributy
require{
type init_t;
type initrc_t;
type tmpfs_t;
type device_t;
type fsadm_log_t;
type udev_tbl_t;
type syslogd_t;
type udev_t;
type lo_node_t;
type local_login_t;
type security_t;
type mount_t;
type fsadm_t;

};

 # Pravidla pro umozneni operaci s adresarem /tmp
 # a ze soubory v nem

allow init_t tmpfs_t:chr_file { ioctl read write };
allow init_t tmpfs_t:dir { add_name search write };
allow init_t tmpfs_t:fifo_file { create getattr read write };
allow initrc_t tmpfs_t:chr_file { ioctl read write };
allow initrc_t tmpfs_t:file { create read write };

 # Veskere operace pro manipulaci se zarizenimi
 # v adresari /dev

allow initrc_t device_t:blk_file { create setattr };
allow initrc_t device_t:chr_file { create setattr };
allow initrc_t device_t:fifo_file setattr;


 dev_manage_all_dev_nodes(initrc_t)
 allow initrc_t udev_tbl_t:dir { add_name create write };
 allow initrc_t udev_tbl_t:dir write;
 allow initrc_t udev_tbl_t:dir { add_name create write };

 # Povoleni zapisu do zaznamenavaciho souboru
 # pro fsck (kontrola disku)

allow initrc_t fsadm_log_t:file write;

 # Povoleni domene intrc_t pojmenovat soket vytovreny
 # typem lo_node_t

allow initrc_t lo_node_t:tcp_socket node_bind;
```

F MODIFIKACE DOSTUPNÝCH PRAVIDEL

```
policy_module(ap2,1.0.0)

require{
type user_t;
type ftpd_t;
};

# Modul vyuzivajici stavajici pravidla pro
# uspesny beh na dane konfiguraci

####### Webovy server Apache #######

# Vytvoreni typu pro webove stranky a cgi skripty
# prostrednictvim templa rozhrani

apache_content_template(xgrepl03)

#
#Vytvori:  httpd_xgrepl03_content_t
# httpd_xgrepl03_script_t

userdom_search_all_users_home_dirs(httpd_t)

allow httpd_t httpd_xgrepl03_script_t:dir r_dir_perms;
allow httpd_t httpd_xgrepl03_script_t:file r_file_perms;

# Nutne aktivovat prislusne boolean pro korektni
# cinnost
# setsebool -P httpd_enable_cgi 1
# setsebool -P httpd_unified 1
# setsebool -P httpd_bultin_scripting 1

########## FTP ##########

### Varianta 1 - nevhodne, neni izolace ###

#umozni domene ftpd_t domene ridit adresare a soubory
#s typy user_home_t, user_home_dir_t

ftp_per_role_template(user)

### Varianta 2 ###

# Deklarace typu
# type xgrepl03_ftpd_home_t;

# Prohledavani domovskych adresaru a souboru
# userdom_search_generic_user_home_dirs(proftpd_t)
# files_search_home(proftpd_t)
# files_getattr_home_dir(proftpd_t)

# Neauditovani pokusu o prohledavani adresaru superuzivatele
# userdom_dontaudit_list_sysadm_home_dirs(proftpd_t)
# userdom_dontaudit_root_home_dirs(proftpd_t)

# Umozneni domene zapisovat, cist do/z souboru a menit obsah adresaru
# allow ftpd_t xgrepl03_ftpd_home_t:file rw_file_perms;
# allow ftpd_t xgrepl03_ftpd_home_t:dir rw_dir_perms;
```

G ADMINISTRACE FTP A HTTP SERVERU

```
policy_module(sshadmin,1.0.0)

# Deklarace externiho atributu

require{
type staff_t;
};

# mozneni prechodu roli user_r do staff_r role
# prechod musi byt implicitne povolen

userdom_role_change_staff(user)

### Administrace weboveho serveru ###

# Pravidla nutna pro restart sluzby
# Povoleni prechodu do domeny weboveho serveru

apache_domtrans(staff_t)
role_transition staff_r httpd_exec_t system_r;
allow staff_r system_r;

# Pravidla pro pristup k jednolivym konfiguracnim
# a systemovym souboru serveru

apache_manage_web_content(staff_t)
apache_read_config_files(staff_t)
apache_read_lock_files(staff_t)
apache_read_run_files(staff_t)

### Administrace ftp serveru ###

# Pravidla nutna pro restart sluzby
# Prechod do domeny proftpd_t

proftpd_domtrans(staff_t)
role_transition staff_r proftpd_exec_t system_r;

# pristup ke konfiguracnim souborum
ftp_read_config_files(staff_t);

# editace uzivatelskych a systemovych
# soboru

ftp_manage_content(staff_t)
ftp_pid_manage(staff_t)

# neauditovani pokusu pro prohledavani adresaru superuzivatele

userdom_dontaudit_search_sysadm_home_dirs(staff_t)
userdom_dontaudit_search_generic_user_home_dirs(staff_t)
```