Slovenská technická univerzita v Bratislave,
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Študijný odbor: INFORMATIKA

Bc. Marek Zelem

Integrácia rôznych bezpečnostných politík do OS Linux

Diplomová práca

---

máj 2001

Vedúci diplomovej práce:
prof. Ing. Jiří Šafařík, PhD.

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Študijný odbor: INFORMATIKA
Autor: Bc. Marek Zelem
Diplomová práca: Intergrácia rôznych bezpečnostných politík do OS Linux
Vedenie diplomovej práce: prof. Ing. Jiří Šafařík, PhD.
máj 2001

Práca sa zaoberá analýzou problematiky bezpečnostných politík a možnosti ich začlenenia do operačného systému Linux. Zvýšená pozornosť je venovaná bezpečnostnému modelu RBAC (Role Based Access Control) a projektu Medusa DS9, ktorý poskytuje prostriedky na rozšírenie bezpečnostnej politiky operačného systému Linux.

V práci sú navrhnuté prostriedky pre začlenenie rôznych bezpečnostných politík do operačného systému Linux na báze projektu Medusa DS9. Definuje jednotný priestor mien systémových objektov a jednotný pohľad na práva s cieľom zabezpečiť spoluprácu rôznych bezpečnostných modelov vrátane RBAC.

Navrhované prostriedky sú implementované ako modulárny autorizačný server s možnosťou ďalšieho rozširovania. Implementácia zahŕňa aj modul zabezpečujúci uplatňovanie a administráciu bezpečnostného modelu RBAC.

Annotation

Slovak University of Technology Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY
Degree Course: INFORMATICS
Author: Bc. Marek Zelem
Thesis: Integration of various security policies into OS Linux
Supervisor: prof. Ing. Jiří Šafařík, PhD.
2001, May

This thesis focuses on the analysis of the issues of security policies and the possibilities of their integration into the Linux operating system. It details the RBAC (Role Based Access Control) security model and the project Medusa DS9 which is suitable to use for extending the security policy of Linux.

As an effort to incorporate various security policies into the Linux operating system, this thesis draws a proposal of the needed resources based on the existing project Medusa DS9. It defines a common namespace of the system objects and the unified permission handling in a manner which ensures the cooperation of various security models including RBAC.

The proposal is implemented as a modular authorization server, ready for the future extensions. The implementation also covers the RBAC enforcement and the administration module.

Prehlásenia

Prehlasujem na svoju česť, že som na diplomovej práci pracoval samostatne, na základe vlastných teoretických poznatkov a skúseností, pričom som ako pomôcku použil uvedenú literatúru.

Bratislava, máj 2001.

Bc. Marek Zelem

Obsah

1 Úvod

Operačný systém Linux uplatňuje bezpečnostnú politiku riadenia prístupu, ktorá nie vždy vyhovuje náročným požiadavkám používateľov a správcov. Existuje množstvo bezpečnostných modelov, ktoré sa snažia týmto požiadavkám čo najlepšie vyhovieť. Pritom žiaden model nie je lepší ako iné, lebo každý je zameraný viac či menej na vyhovenie tej ktorej požiadavke. Výber bezpečnostného modelu preto závisí od kladených požiadaviek. Požiadavky môžu byť navyše natoľko rôznorodé, že im žiaden bezpečnostný model dostatočne nevyhovuje. V takom prípade je vhodné použiť niekoľko bezpečnostných modelov súčasne, pritom je potrebné zabezpečiť aby tieto modely spolupracovali. Aby toto bolo možné, je potrebné do operačného systému začleniť mechanizmy, ktoré umožnia uplatňovanie rôznych bezpečnostných politík.

Integrácia rôznych bezpečnostných politík do operačného systému Linux teda predstavuje problém, ktorý je možné rozdeliť na dve časti. Prvou časťou je vnútenie bezpečnostnej politiky do operačného systému Linux, ktorý už uplatňuje svoju bezpečnostnú politiku. Druhou časťou je problém spolupráce rôznych bezpečnostných politík. Táto práca sa zaoberá analýzou, návrhom riešenia a implementáciou oboch týchto problémov.

Práca analyzuje existujúce použiteľné prostriedky na vnucovanie bezpečnostnej politiky do operačného systému Linux a navrhuje riešenie, ktoré využije tieto prostriedky na začlenenie rôznych bezpečnostných politík do operačného systému Linux. Práca sa zaoberá tiež implementáciou tohto riešenia.

2 Analýza problematiky

2.1 Bezpečnostná politika

Cieľom bezpečnostnej politiky riadenia prístupu je určiť, ktoré činnosti je v systéme možné a za akých okolností vykonať. Vo všeobecnosti je možné na všetky činnosti pozerať ako na operácie vykonávane nejakým subjektom nad nejakým objektom. Subjekt predstavuje takzvaný aktívny objekt, teda objekt, ktorý iniciuje (požaduje od systému) vykonanie operácie.

V konečnom dôsledku teda bezpečnostná politika rozhoduje, či trojica (operácia, subjekt, objekt) bude systémom vykonaná alebo zamietnutá z dôvodu neoprávnenosti subjektu vykonať žiadanú operáciu nad objektom.

2.2 Bezpečnostný model

Určovať bezpečnostnú politiku na základe trojíc (operácia, subjekt, objekt) by bolo veľmi rozsiahlé a neprehľadné. Navyše nie je vždy možné identifikovať všetky subjekty a objekty ešte pred ich vznikom, takže by napríklad nebolo možné určiť či nejaký ešte neexistujúci subjekt bude môcť vykonať nejakú operáciu nad nejakým objektom.

Bezpečnostný model definuje dalšie prvky, ako napríklad bezpečnostné úrovne, oddelenia, role a pod. Pomocou týchto dalších prvkov a ich asociácie s objektami a subjektami je možné jednoduchšie a hlavne prehľadnejšie definovat oprávnenia, teda v konečnom dôsledku množinu trojíc (operácia, subjekt, objekt).

Bezpečnostná politika definovaná pre nejaký bezpečnostný model potom neurčuje trojice (operácia, subjekt, objekt) priamo, ale nepriamo pomocou prvkov definovaných bezpečnostným modelom.

Bezpečnostný model teda často vyžaduje, aby objekty a subjekty pri sebe niesli prídavné informácie, ktoré ich asociujú s modelom definovanými prvkami. Takéto prídavné informácie sa niekedy označujú ako bezpečnostné nálepky (security labels).

2.3 Prístupová matica

Množina všetkých trojíc (operácia, subjekt, objekt) sa dá prepísať do matice, kde riadky predstavujú subjekty a stĺpce predstavujú objekty. Prvky matice obsahujú množinu operácií, ktoré predstavujú právo subjektu odpovedajúcemu danému riadku vykonávať tieto operácie nad objektom odpovedajúcim danému stĺpcu. Takáto matica sa nazýva prístupová matica. Všetky varianty prístupových matíc vychádzajú z modelu, ktorý predstavil Lampson [LAM71] a Graham a Denning [GD72].

2.4 Bezpečnostný model systému UNIX

Bezpečnostný model systému UNIX vychádza z abstrakcie súborov, kde všetky pasívne objekty systému UNIX sú reprezentované súbormi. Toto bola pôvodná koncepcia. Postupom času bol však UNIX rozširovaný o ďalšie prvky, ako napríklad sieť a mechanizmy medziprocesovej komunikácie. Tieto prvky žiaľ už nie sú reprezentované súbormi a tak porušujú pôvodnú jednoduchú a jednotnú koncepciu.

Bezpečnostný model systému UNIX definuje používateľov a skupiny. Používateľ reprezentuje jedného individuálneho používateľa. Skupiny môžu zahŕňať viacerých používateľov.

Každý súbor má priradeného používateľa a skupinu. Používateľ sa často označuje ako vlastník. Obyčajne je to používateľ, ktorý súbor vytvoril. Súbor ma okrem vlastníka a skupiny definované aj prístupové práva. Tieto sa delia na tri časti:

1. Práva vlastníka (owner)
2. Práva skupiny (group)
3. Práva ostatných (others)

Každé toto právo sa skladá z troch častí odpovedajúcich trom typom prístupu:

1. Čítanie (read)
2. Zápis (write)
3. Vykonanie (execute)

Procesy (jediné aktívne objekty) majú asociovaného používateľa, efektívneho používateľa a skupiny. Používateľ je ten kto program spustil a obyčajne je zhodný s efektívnym používateľom. Efektívny používateľ a skupiny reprezentujú práva procesu. Povolený je len taký prístup k súboru, ktorý zodpovedá právam určeným pri súbore pre:

1. vlastníka, ak efektívny používatľ procesu je zhodný s vlastníkom súboru
2. skupinu, ak sa skupina súboru zhoduje s jednou zo skupín procesu
3. ostatných

Pritom platí, že sa vyhodnocuje len jeden z uvedených bodov, takže napríklad skupina môže mať menšie práva ako ostatní. Meniť prístupové práva môže iba vlastník súboru.

Vykonanie súboru je činnosť pri ktorej proces nahradí svoj program programom, ktorý je zapísaný v súbore. Vykonanie adresára predstavuje právo prechádzať týmto adresárom.

Okrem týchto práv existujú ešte príznaky SUID a SGID. Ak sa vykonáva súbor, ktorý má príznak SUID, tak proces zmení svojho efektívneho používateľa na vlastníka súboru, tým získa jeho práva. Podobne je to aj s príznakom SGID, ktorý sa týka skupiny.

V systéme UNIX existuje privilegovaný používateľ - správca. Tento správca má tradične identifikačné číslo (UID) 0 a volá sa „root". Má právo vykonávať všetky operácie nad všetkými súbormi. Okrem iného má právo meniť prístupové práva všetkých súborov a tiež má právo meniť vlastníka súboru.

Všetky ostatné operácie v systéme nevzťahujúce sa na súbory je možné rozdeliť na dve skupiny: privilegované a neprivilegované. Neprivilegované smie vykonávať každý, privilegované len root. Ide najmä o správcovské operácie meniace rôzne nastavenia jadra operačného systému, ako napríklad zmena systémového času, alebo konfigurácia siete a podobne.

2.5 Bezpečnostný model systému Linux

Operačný systém Linux vychádza z koncepcie systému UNIX. Preto má aj veľmi podobný bezpečnostný model. Rozšírený je len o koncept tzv. POSIX capability (capabilities). Capability predstavujú čiastočné práva správcu, čím sú vlastne rozdelené neobmedzené správcovské práva na niekoľko častí. Delenie je žiaľ dosť nerovnomerné a niektoré capability sú tak silné, že sa pomocou nich dajú nepriamo získať aj všetky ostatné.

Capability sa v zásade delia na niekoľko skupín:

1. Capability vzťahujúce sa na súborový systém. Tu patrí napríklad aj právo meniť vlastníka súborov alebo právo pristupovať k všetkým súborom bez ohľadu na ostatné práva.
2. Capability vzťahujúce sa na procesy. Tu patrí právo posielať signál ľubovoľnému procesu, právo meniť svoj uid a gid, právo meniť svoje capability.

3. Capability vzťahujúce sa na sieť. Tu patrí právo používať porty sieťových protokolov TCP a UDP menšie ako 1024, právo prijímať a vysielať RAW pakety a rôzne práva administrácie siete.
4. Capability vzťahujúce sa k medziprocesovej komunikácii (IPC).
5. Capability na administráciu systému a rôzne špeciálne práva ako napríklad vkladanie modulov do jadra, zmena koreňového adresára pre proces, právo vypínať a reštartovať systém, právo meniť systémový čas a podobne.

POSIX-ové capability majú aj niektoré dobré vlastnosti. Umožňujú, aby aj iný používateľ ako root mohol robiť niektoré, nie veľmi nebezpečné, operácie. Takto je možné rozdeliť úlohu jedného správcu na viacero častí, ktoré môžu v systéme plniť rôzni používatelia. Iným príkladom je, že správca systému alebo jeden zo správcov, ktorý má samozrejme svoje vlastné používateľské konto a obyčajne nepracuje v systéme ako root, môže bez toho aby sa prihlasoval ako root, zrušiť zaťažujúce procesy, ktoré nevlastní alebo im môže zmeniť prioritu.

Z bezpečnostného hľadiska je nutné posúdiť ako je možné pomocou capabilít ohroziť bezpečnosť systému. Niektoré capability dávajú schopnosť ohroziť systém iba v zmysle narušenia jeho normálneho chodu, teda zneprístupnenia služieb. Iné zas poskytujú schopnosť, ktorá sama o sebe systém neohrozí, ale umožňuje aby sa dala zneužiť na získanie ďalších práv. To je napríklad právo používať porty menšie ako 1024, alebo právo prijímať a posielať RAW pakety. Medzi capabilitami sú aj také, ktoré predstavujú tak silné schopnosti, že prípadnému narušiteľovi postačí len jedna takáto capabilita. Takou capabilitou je napríklad právo vkladať moduly do jadra alebo právo prístupu a hlavne zápisu do ľubovoľného súboru.

Z uvedeného je zrejmé, že POSIX-ové capability predstavujú veľmi nerovnomerné rozdelenie práv, čo je veľmi záporná skutočnosť. Ďalšou zápornou vlastnosťou je fakt, že capability predstavujú právo, ktoré sa vzťahuje na všetky objekty. Nie je teda možné mať jednu capabilitu, ktorá platí len na jeden alebo niekoľko objektov. Každá capabilita platí vždy pre všetky objekty.

Praktické použitie capabilít v Linuxe je dosť obmedzené vzhľadom na spomínané záporné vlastnosti. Ďalší problém použitia capabilít je skutočnosť, že ich nepodporujú žiadne systémové programy. Podpora capabilít v jadre síce je, ale na zabezpečenie spätnej kompatibility jadro obsahuje emuláciu správania sa systému UNIX pri spúšťaní súborov s nastaveným SUID príznakom. Prakticky to znamená, že ak je spustený súbor vlastnený root-om s SUID príznakom, tak proces získava automaticky aj všetky capability okrem jedinej - práva dávať capability iným procesom.

Štandardne tiež neexistuje žiadny používateľsky pohodlný spôsob získavania capabilít. Existuje niekoľko projektov, ktoré sú zamerané na využitie Linuxovým jadrom podporovaných capabilít. Jedným z týchto projektov je úprava ELF formátu spustiteľných súborov [MACH2000] tak aby umožňovali pri spustení súboru s nastaveným SUID príznakom určiť, ktoré capability má tento program získať. Za normálnych okolností by takýto program získal všetky capability. Príkladom použitia je program ping, ktorý na svoju činnosť potrebuje len jednu capabilitu - právo vysielať RAW pakety. Preto nemusí mať aj všetky ostatné capability. Tým sa znižuje možnosť narušenia systému narušením programu ping.

2.6 Bezpečnostný model RBAC

Koncept riadenia prístupu založeného na rolách (RBAC) [FCK95][SAN96] začal vznikať spolu s multipoužívateľskými systémami v 70-tých rokoch 20. storočia. Základnou myšlienkou RBAC je to, že práva sú priradené rolám a nie používateľom. Používatelia sú potom priradení jednotlivým rolám. To výrazne zjednodušuje správu RBAC modelu. Role sú vytvárané tak, aby zodpovedali existujúcim funkciám v organizácii. Používatelia (zamestnanci) vykonávajúci tieto funkcie sú jednoducho priradení zodpovedajúcim rolám. Rolám sú priradené iba práva potrebné na výkon tej ktorej funkcie. Keď je nejaký človek v organizácii preradený na inú funkciu, tak mu jednoducho stačí zmeniť rolu. Podobne ak je do systému pridaná nová aplikácia, stačí príslušne modifikovať práva rolí. Vo všeobecnosti je potrebné práva rolí meniť menej často ako priradenie používateľov rolám [NIST93].

2.6.1 Role a skupiny

Mnoho systémov riadenia prístupu umožňuje vytvárať skupiny používateľov. Hlavným rozdielom medzi konceptom skupín a konceptom rolí je, že skupiny typicky predstavujú zoskupenie používateľov a nie zoskupenie práv. Rola je aj zoskupenie používateľov aj zoskupenie práv.

V systéme UNIX je členstvo v skupinách definované v súboroch /etc/passwd a /etc/group. Preto je jednoduché definovať členstvo používateľov v skupinách modifikovaním týchto dvoch súborov. Práva skupín sú však definované pri každom súbore. Je teda výrazne jednoduchšie definovať členstvo v skupine ako práva skupiny. Navyše je priradenie práv skupine silne decentralizované, nakoľko vlastník súboru môže sám určovať skupinu do ktorej súbor patrí, ak je jej členom. Vlastník môže tiež modifikovať práva skupiny na svoj súbor. Role majú členstvo definované taktiež centralizovane. Práva rolí sú ale na rozdiel od práv skupín definované centrálne.

Role teda zodpovedajú dvom charakteristikám:

1. Je približne rovnako ťažké definovať členstvo ako práva.
2. Správa členstva role ako aj správa práv role je centralizovaná.

2.6.2 Štyri triedy modelov RBAC

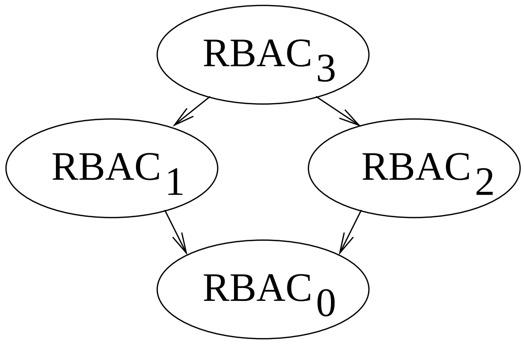
Konceptuálny model RBAC bol postupne vyvíjaný a rozširovaný o ďalšie komponenty, ktoré predstavujú doplnenie základného modelu. Pre prehľadnosť boli definované štyri triedy modelov RBAC [SAN96].

RBAC$_0$ - Základný model. Predstavuje minimum, čo musí obsahovať každý systém riadenia prístupu založený na rolách.

RBAC$_1$ - Rozšírený model. Zahŕňa RBAC$_0$ a pridáva hierarchiu rolí. Teda umožňuje, aby jedna rola dedila všetky práva inej role.

RBAC$_2$ - Rozšírený model. Zahŕňa RBAC$_0$ a pridáva množinu obmedzení. Obmedzenia predstavujú logické podmienky, ktoré sú aplikované na všetky časti RBAC.

RBAC$_3$ - Zjednotený model. Zahŕňa RBAC$_1$ a RBAC$_2$.



Obr. 1: Vzájomný vzťah medzi triedami modelov RBAC

2.6.2.1 Základný model RBAC$_0$

Obsahuje tri množiny entít:

U - Používatelia (Users)

R - Role (Roles)

P - Oprávnenia (Permissions)

Entita používateľ nemusí predstavovať len človeka, môže to byť aj iná aktívna entita, napríklad bežiaci proces zabezpečujúci nejakú systémovú službu.

Entita oprávnenie predstavuje oprávnenie vykonať nejakú operáciu nad jedným, alebo viacerými objektami v systéme. Oprávnenie je vždy pozitívne. Teda oprávňuje, nezakazuje. Pritom platí, že čo nie je povolené, je zakázané. Charakter oprávnení silne závisí od systému a od implementačných detailov.

Oprávnenie môže byť špecifické, ako napríklad oprávnenie na čítanie časti súboru, alebo všeobecné, ako napríklad oprávnenie na čítanie všetkých súborov. Oprávnenie môže tiež zahŕňať viacero operácií, napríklad presun údajov z jedného súboru do iného.

$RBAC_0$ okrem spomínaných troch množín entít obsahuje aj nasledovné relácie medzi týmito entitami.

UA - Priradenie používateľov (User Assignment). Je to relácia typu N-N medzi množinou používateľov a množinou rolí, ktorá určuje členstvo používateľov v rolách.

PA - Priradenie oprávnení (Permission Assignment). Relácia typu N-N medzi množinou oprávnení a množinou rolí, ktorá definuje oprávnenia jednotlivých rolí.

S - Prihlásenie (Session). Relácia typu 1-N medzi jedným používateľom a viacerými rolami. Používateľ pri prihlásení sa do systému aktivuje ľubovoľnú podmnožinu svojich rolí. Táto podmnožina je platná len pre jedno konkrétne prihlásenie. Aktuálne práva predstavujú zjednotenie práv aktivovaných rolí. Používateľ môže byť v jednom čase viackrát prihlásený. $RBAC_0$ tiež umožňuje, aby používateľ dynamicky aktivoval a deaktivoval role počas jedného prihlásenia.

2.6.2.2 Hierarchia rolí - $RBAC_1$

$RBAC_1$ navyše oproti $RBAC_0$ obsahuje reláciu medzi rolami navzájom, ktorá vyjadruje dedenie práv. Príkladom použitia hierarchie rolí je pracovný vzťah podriadený - nadriadený pracovník. Nadriadený pracovník je oprávnený robiť všetko to, čo jeho podriadený, plus má navyše nejaké ďalšie oprávnenia.

2.6.2.3 Obmedzenia - $RBAC_2$

Oproti $RBAC_0$ obsahuje $RBAC_2$ navyše množinu obmedzení. Tieto obmedzenia sa vzťahujú na relácie UA, PA a S a na množiny entít U, R a P. Obmedzenia sú predikáty, ktoré hovoria, či sú hodnoty rôznych častí RBAC "akceptovateľné" alebo "neakceptovateľné". Povolené sú len akceptovateľné hodnoty. Z implementačných dôvodov by mali byť výroky obmedzení čo najjednoduchšie, aby sa dali čo najefektívnejšie vyhodnotiť.

Najčastejším obmedzením v RBAC_2 je požiadavka na vzájomné vylučovanie rolí. Teda že jeden používateľ môže byť najviac v jednej role z množiny vzájomne sa vylučujúcich rolí. Rozlišujú sa dva typy tohto obmedzenia. Statické (SSD) vylučuje aby boli jednému používateľovi priradené vzájomne sa vylučujúce role. Dynamické (DSD) vylučuje aby bol jeden používateľ v dvoch vylučujúcich sa rolách súčasne. Pri DSD ide prakticky skôr o obmedzenie na reláciu prihlásenie, kdežto pri SSD ide o obmedzenie na reláciu priradenie používateľov.

Ďalšie časté obmedzenia sú kardinalitné obmedzenia. Rola môže mať stanovený maximálny alebo minimálny počet používateľov, ktorí môžu byť jej členmi. Alebo tiež používateľ môže mať stanovený maximálny alebo minimálny počet rolí, v ktorých môže byť.

Ďalším užitočným obmedzením je obmedzenie, že sa používateľ môže stať členom jednej role A iba ak je už členom role B. Pre konzistenciu môže byť vyžadované, aby role bolo možné priradiť právo P len ak je jej už priradené právo Q.

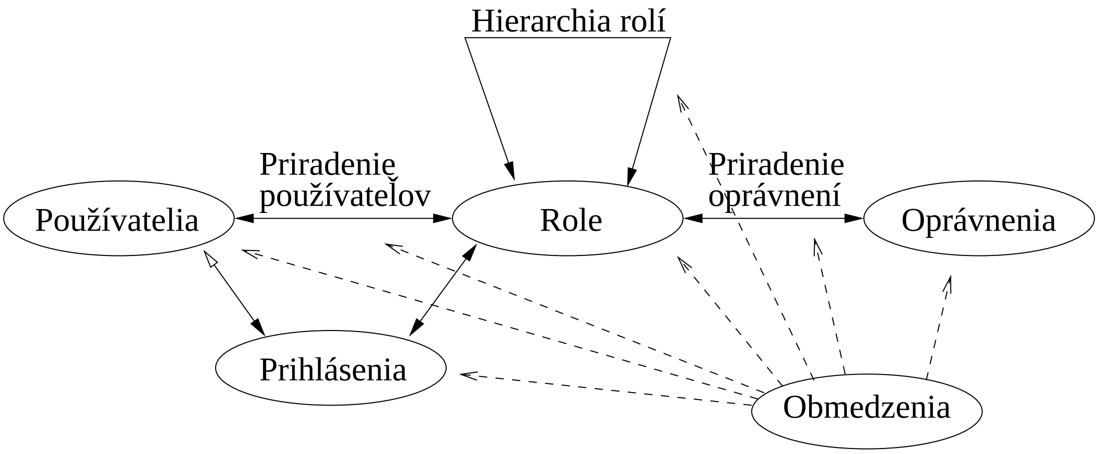
Podobné obmedzenia môžu byť aplikované nielen na reláciu priradenia používateľov ale aj na reláciu prihlásenie (session).

Obmedzenia relácie priradenia používateľa vyžadujú, aby jeden človek vystupoval iba ako jeden používateľ. Inak by mohol niektoré obmedzenia obísť. Podobne je to s oprávneniami. Povolenie jednej operácie by nemalo byť súčasťou dvoch rôznych oprávnení. Toto musí zabezpečiť správca systému pri jeho konfigurácii.

2.6.2.4 Zjednotený model - $RBAC_3$

$RBAC_3$ predstavuje zjednotenie $RBAC_1$ a $RBAC_2$. Teda obsahuje aj hierarchiu rolí aj množinu obmedzení. Pri tejto kombinácii sa vynárajú problémy, ktorých riešenia sú hľadané v rámci RBAC komunity a nemá význam ich tu rozoberať.

Úplný model $RBAC_3$ znázorňuje obrázok 2.



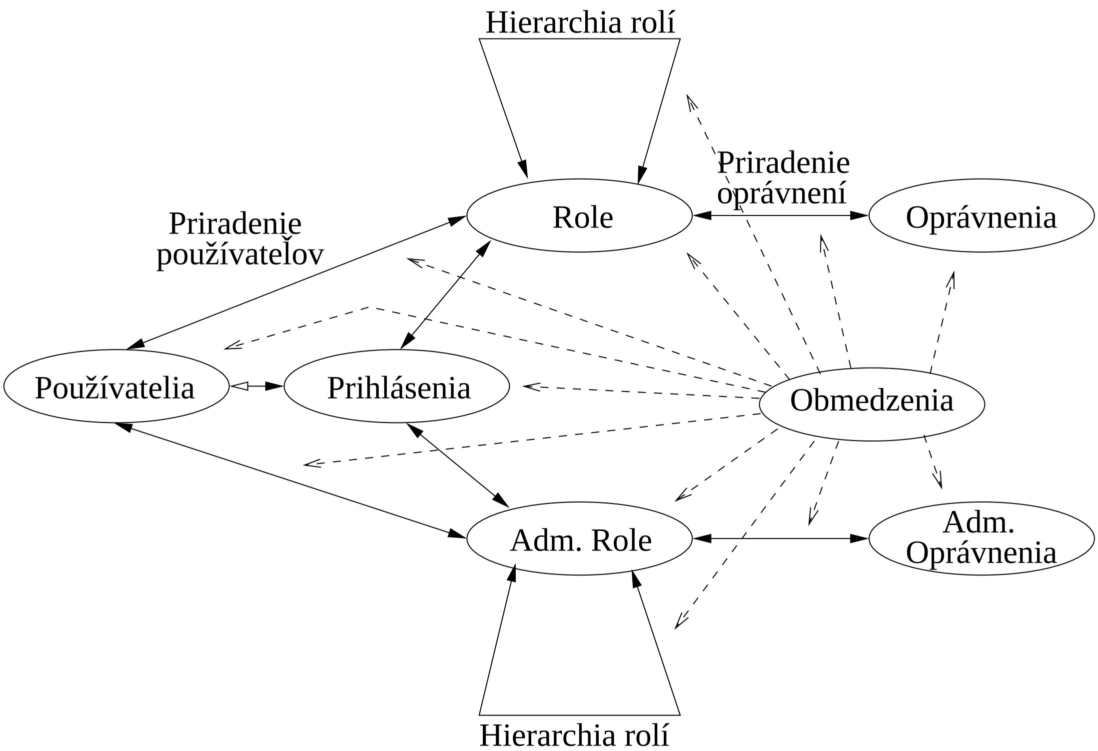
Obr. 2: Model $RBAC_3$

2.6.3 Správa RBAC

Na správu RBAC sa môžeme pozerať ako na správu relácií a množín. Teda správu priradenia používateľov, správu priradenia práv a správu hierarchie rolí. Tiež je potrebné spravovať aj množinu používateľov, množinu rolí, množinu oprávnení a množinu obmedzení.

Kombinácia hierarchie rolí a obmedzení umožňuje decentralizovať správu rolí a hierarchie rolí. Najvyšší bezpečnostný správca môže nastavením vhodných obmedzení limitovať pôsobnosť správcov rolí.

Podrobne sa problematikou správy RBAC modelu zaoberá model ARBAC97 [ARB97] a jeho nasledovník ARBAC99 [ARB99]. ARBAC okrem iného definuje popri oprávneniach aj špeciálne administratívne oprávnenia, ktoré dávajú ich držiteľovi právo na modifikáciu príslušných častí RBAC. Taktiež definuje pojem administratívna rola. Toto je znázornené na obrázku 3.



Obr. 3: Model ARBAC

2.7 Uplatňovanie bezpečnostnej politiky

Operačný systém poskytuje služby, ktoré predstavujú vykonanie požadovaných operácii. Pred ich vykonaním musí rozhodnúť, či sa vykonať smú alebo nie. Rôzne operačné systémy na to používajú rôzne mechanizmy. Taktiež väčšina operačných systémov uplatňuje nejaký bezpečnostný model. Často však len jeden konkrétny. Tak je to napríklad aj v operačnom systéme Linux, ako aj vo všetkých operačných systémoch typu UNIX.

Operačný systém Linux vychádza z monolitickej podoby systému UNIX. Pritom ale má modulárnu štruktúru. Uplatňovanie bezpečnostnej politiky je žiaľ rozptýlené po veľkej časti jadra operačného systému. To značne komplikuje zmenu bezpečnostného modelu. Ďalšou komplikáciou je množstvo používaných systémových programov, ktoré sa spoliehajú na bezpečnostný model systému Linux.

Existuje niekoľko projektov zaoberajúcich sa doplnením, alebo zmenou bezpečnostného modelu systému Linux. Všetky projekty zachovávajú aj prvky Linuxového modelu aby nevyžadovali zmenu veľkého množstva systémových programov. Za zmienku stoja projekty, ktoré definujú bezpečnostné rozhranie medzi jadrom operačného systému Linux a bezpečnostným rozhodovacím centrom (Security decision center) alebo autorizačným serverom. Jedine takéto projekty môžu poskytnúť prostriedky na vnútenie externe definovanej bezpečnostnej politiky do operačného systému.

2.7.1 Projekt RSBAC

Projekt RSBAC (Rule Set Based Access Control) je zameraný na zvýšenie bezpečnosti operačného systému Linux [RSB2000]. Vykonáva kontrolu bezpečnostne relevantných volaní jadra. Celý bezpečnostný systém sa nachádza v jadre operačného systému. Dobrou vlastnosťou je jeho modulárnosť, lebo umožňuje pridanie rozhodovacích modulov uplatňujúcich rôzne bezpečnostné modely. Pre rozhodovacie moduly definuje rozhranie GFAC (Generalized Framework for Access Control).

Tento bezpečnostný systém postihuje súborový systém, vstupno/výstupné zariadenia, objekty medziprocesovej komunikácie a používateľov. Rozhodnutia o povolení alebo nepovolení prístupu vykonávajú rozhodovacie moduly na základe typu prístupu, objektu na ktorý sa pristupuje a atribútov.

Nevýhodou tohto projektu je, že sa celý bezpečnostný systém nachádza v jadre operačného systému, takže rozhodovacie moduly nemôžu, tak ako používateľské procesy, využívať pohodlné služby jadra napríklad na prácu so súbormi.

2.7.2 Projekt Security-Enhanced Linux (NSA)

Projekt Security-Enhanced Linux je založený na bezpečnostnej architektúre **Flask** [SPE99], ktorá bola navrhnutá už skôr pre operačný systém Fluke.

Flask je sada mechanizmov slúžiacich na zabezpečenie riadenia prístupu. Sleduje interakcie medzi objektami v systéme a robí rozhodnutia. Skladá sa z dvoch častí:

1. vnucovanie bezpečnostnej politiky (Enforcement) - Object manager
2. samotná bezpečnostná politika (Policy) - Security server

Object manager sleduje všetky dôležité interakcie a posiela ich na rozhodnutie bezpečnostnému serveru (Security server). Na dosiahnutie čo najväčšieho výkonu sú rozhodnutia dočasne uložené v cache a ak nastane rovnaká situácia, tak sa použije odpoveď z cache.

2.7.3 Projekt Medusa DS9

Projekt Medusa DS9 [ZPO2000] vychádza z ročníkového projektu študentov FEI STU v Bratislave Mareka Zelema a Milana Pikulu. Projekt mal názov "Zvýšenie bezpečnosti OS Linux" [ZP99]. Po jeho obhájení ako ročníkového projektu autori pokračovali v jeho dalšom vývoji vo svojom voľnom čase.

Medusa DS9 sa skladá z dvoch logických častí, ktoré dohromady tvoria jeden celok:

1. VS monitor
2. Autorizačný server

2.7.3.1 Model VS monitoru

Model VS monitoru, nazývaný VS model, je možné definovať na základe prístupovej matice [LAM71][GD72], pričom táto je dekomponovaná na dve časti:

1. schopnosti subjektov[1] - asociované s každým subjektom
2. príslušnosť objektov - asociovaná s každým objektom

Každý objekt má definovanú príslušnosť k virtuálnym svetom (VS - Virtual Space). Jeden objekt môže byť súčasne v jednom, alebo vo viacerých virtuálnych svetoch. Prakticky nemusí byť v žiadnom, ale tento stav v praxi nemá veľké uplatnenie. Príslušnosť objektu k virtuálnym svetom je množina virtuálnych svetov $\mathbf{OVS(o)}$, do ktorých objekt patrí.

Každý subjekt má definované schopnosti. Tieto sú reprezentované množinou virtuálnych svetov $\mathbf{SVS_a(s)}$ pre každý typ operácie $\boldsymbol{a}$ (typ prístupu). Subjekt $\boldsymbol{s}$ má právo vykonať operáciu typu $\boldsymbol{a}$ nad objektom $\boldsymbol{o}$ vtedy a len vtedy, ak je prienik schopností subjektu $\boldsymbol{s}$ pre operáciu typu $\boldsymbol{a}$ s príslušnosťou objektu $\boldsymbol{o}$ neprázdny, teda ak $\mathbf{SVS_a(s)} \cap \mathbf{OVS(o)} \neq \emptyset$.

Medusa DS9 rozlišuje tri typy operácií nad objektami:

1. Čítanie (Read) - čítanie informácii z objektu
2. Zápis (Write) - zápis informácií do objektu
3. Videnie (See) - schopnosť zistiť existenciu objektu

V súlade so systémom UNIX je vytváranie súborov v adresári považované za zápis do adresára. Vykonávanie súborov VS monitor nepostihuje, rozhodnutie

[1] Nejedná sa o capability. Anglický termín pre tieto schopnosti je ability a nemajú priamy súvis s capabilitami.

je teda ponechané na bezpečnostný mechanizmus systému Linux. Vykonávanie súborov je ale možné postihnúť autorizačným serverom.

Keďže aj subjekty často vystupujú v úlohe objektov, teda je nad nimi vykonávaná nejaká operácia, majú ako objekty definovanú aj svoju príslušnosť.

VS model implementovaný v Meduse DS9 funguje v súčinnosti s bezpečnostným mechanizmom operačného systému Linux. Dokáže zakázať vykonanie operácie ale neumožňuje povoliť vykonanie operácie, ktoré by bolo v rozpore s bezpečnostným rozhodnutím Linuxu. Na zamietnutie operácie teda stačí zamietnutie VS modelom alebo Linuxovým mechanizmom. Na povolenie je potrebné povolenie oboch.

2.7.3.2 Vlastnosti VS modelu

VS model má dve veľké výhody. Množiny virtuálnych svetov sú reprezentované bitovým poľom. Každý bit predstavuje jeden virtuálny svet. Jeho hodnota 1 znamená jeho prítomnosť v množine. Má to síce nevýhodu limitovanosti počtu virtuálnych svetov, ale v praxi je vždy počet použitých virtuálnych svetov konečný. Počet virtuálnych svetov je nastaviteľný. Test neprázdnosti prieniku je realizovaný operáciou AND, ktorej prevedenie je na bežných procesoroch rýchle. Ak sa výsledok bitového AND-u nerovná nule, znamená to, že prienik oboch množín je neprázdny a teda operácia bude povolená.

Druhou výhodou je rýchlosť a efektínosť uplatňovania bezpečnostnej politiky založenej na VS modeli, lebo sa uplatňuje priamo na miestach, kde je potrebné rozhodnúť, a nie je nutné o rozhodnutie žiadať autorizačný server. To bolo aj jedným z hlavných návrhových cieľov projektu Medusa DS9.

2.7.3.3 Autorizačný server

Autorizačný server v projekte Medusa DS9 plní dve úlohy:

1. inicializuje a modifikuje príslušnosť objektov a schopnosti subjektov
2. rozhoduje o povolení, alebo zamietnutí operácie tam, kde je VS model z nejakého dôvodu nepostačujúci

Na rozdiel od VS monitoru, ktorý je implementovaný v jadre operačného systému, autorizačný server môže byť mimo jadra, čo mu poskytuje väčší komfort, najmä čo sa týka alokácie pamäte a práce so súbormi. S jadrom operačného systému komunikuje prostredníctvom špeciálneho znakového zariadenia.

Jadro sa na autorizačný server obracia s dvoma typmi žiadostí:

1. žiadosť o inicializáciu príslušnosti objektov
2. žiadosť o schválenie operácie

Každý objekt má okrem svojej príslušnosti definované aj to, ktoré operácie nad ním vyžadujú schválenie autorizačného servera. Jadro žiada len o schválenie týchto operácií. Autorizačný server môže vo všeobecnosti rozhodnúť jednou zo štyroch možností:

Allow - definitívne povolenie operácie bez ohľadu na bezpečnostnú politiku UNIXu

Deny - definitívne zakázanie operácie

Skip - nevykonať operáciu ale jadro povie žiadateľovi, že operácia prebehla úspešne

OK - prenechanie rozhodnutia na bezpečnostú politiku UNIXu

V niektorých prípadoch majú Allow a OK rovnaký význam. Možnosť Skip nie je tiež pre všetky operácie možná. Vyplýva to z povahy operácie, ale aj z implementačných obmedzení projektu Medusa DS9.

Okrem rozhodnutia môže autorizačný server modifikovať schopnosti subjektu, ktorý operáciu požadoval. Môže modifikovať aj schopnosti iných subjektov.

Pre niektoré operácie je možné presmerovať objekt operácie na iný objekt. Presmerovanie napríklad umožňuje, aby rôzne procesy videli namiesto súboru /etc/passwd iné súbory. Bude sa im teda javiť, že súbor /etc/passwd má iný obsah.

2.7.3.4 Komunikačné rozhranie

Jadro operačného systému komunikuje prostredníctvom špeciálneho znakového zariadenia s autorizačným serverom. Komunikačný protokol rozlišuje oba smery komunikácie podľa toho, ktorá strana komunikáciu iniciovala. Pritom vždy platí model „otázka a odpoveď".

Jadro sa na autorizačný server obracia so žiadosťou o schválenie operácie. V princípe ide skôr o informovanie autorizačného servera o výskyte nejakej udalosti (event) v systéme. Na túto skutočnosť má potom autorizačný server možnosť reagovať tak, ako bolo popísané vyššie.

Jadro vo svojej žiadosti posiela informácie o subjekte, ktorý operáciu požaduje. Ide o základné údaje potrebné na jeho identifikáciu a údaje o jeho právach, množiny VS modelu a rozširujúce informácie, ktoré sú určené na použitie autorizačným serverom (tieto údaje sa nazývajú cinfo). Jadro posiela aj informácie o objekte, nad ktorým má byť vykonaná požadovaná operácia. Ide o informácie podobného typu ako informácie o subjekte. Niektoré operácie však môže subjekt vykonávať len sám nad sebou, na tieto operácie sa dá nahliadať ako na unárne operácie. V takom prípade sa informácie o objekte operácie neprenášajú, nakoľko subjekt a objekt operácie sú totožné. Väčšina operácií má svoje ďalšie atribúty, ktoré sa taktiež prenášajú. Príkladom je operácia poslanie signálu. Jej atribútom je číslo signálu.

Autorizačný server na žiadosť jadra odpovedá jednou zo štyroch vyššie opísaných odpovedí. Okrem toho posiela jadru naspäť aj informácie o subjekte. Jadro podľa nich modifikuje svoje informácie. Špeciálnym prípadom je žiadosť jadra o inicializáciu bezpečnostných informácií objektu. Toto je považované za operáciu bez subjektu. Odpoveďou na ňu sú informácie o objekte namiesto informácií o subjekte.

Autorizačný server môže tiež sám iniciovať komunikáciu s jadrom. Môže napríklad žiadať informácie o ľubovoľnom procese. Môže tiež sám poslať nové informácie o ľubovoľnom procese. Takto má možnosť pri výskyte nejakej operácie ovplyvniť nielen proces žiadajúci operáciu ale aj iné procesy, napríklad rodičovský proces.

2.7.3.5 Výhody projektu Medusa DS9

Projekt Medusa DS9 umožnuje aby autorizačný server, ktorý nemusí byť súčasťou jadra operačného systému, uplatňoval v systéme rôzne bezpečnostné politiky a rôzne bezpečnostné modely. Pritom využitie VS modelu a autorizácie len potrebných operácií umožňuje dosiahnúť čo najväčšiu rýchlosť a efektívnosť.

3 Špecifikácia požiadaviek

Je potrebné navrhnúť a vytvoriť prostriedky pre začlenenie rôznych bezpečnostných politík do operačného systému Linux. Navrhnutý a implementovaný systém musí byť schopný uplatňovať bezpečnostnú politiku nad všetkými dôležitými subsystémami operačného systému.

Medzi požiadavky patrí aj návrh jednotného pomenovania objektov všetkých subsystémov, čo umožní spoluprácu rôznych bezpečnostných modelov. Objekty musia tvoriť jednotnú stromovú hierarchickú štruktúru, preto je potrebné zaviesť stromovú hierarchiu aj medzi objektami, ktoré v operačnom systéme Linux stromovú hierarchiu netvoria (IPC objekty), prípadne tvoria, ale táto nie je úplne vyhovujúca z hľadiska pohodlného opisu bezpečnostného modelu (procesy).

Keďže riešený problém je pomerne všeobecný, navrhnutý systém by mal byť čo najuniverzálnejší, aby ho bolo možné použiť aj na inom operačnom systéme, ako je OS Linux. Nemal by teda byť závislý na konkrétnych typoch objektov existujúcich v operačnom systéme Linux. Mal by byť navrhnutý tak, aby bolo jednoduché doplniť doň podporu ďalších subsystémov a ďalších typov objektov.

Pri implementácii by mal byť navyše kladený dôraz aj na efektívnosť a čo najmenšiu réžiu uplatňovania bezpečnostných politík. Implementované by mali byť mechanizmy zabezpečujúce súbežné uplatňovanie rôznych bezpečnostných modelov, vrátane modelu RBAC.

Návrh a implementácia musí obsahovať aj prostredky na administráciu RBAC modelu, na ktorú sa tiež vzťahuje uplatňovanie bezpečnostnej politiky.

4 Návrh

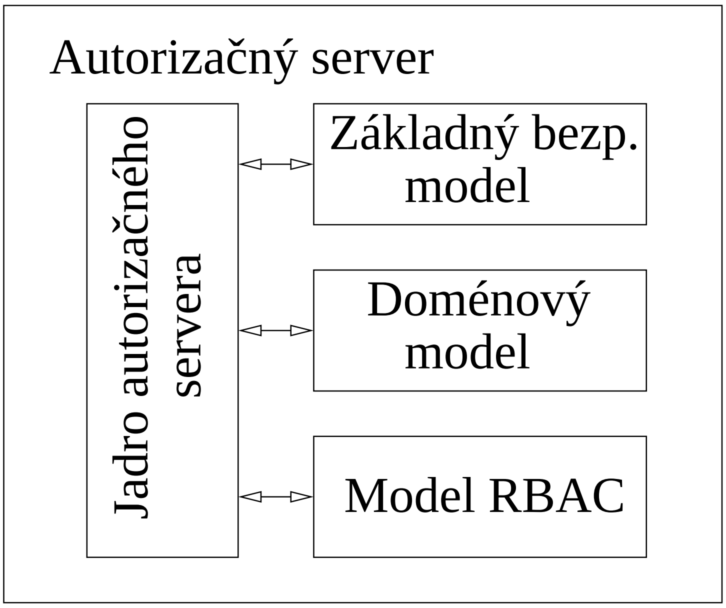
Začlenenie rôznych bezpečnostných politík do operačného systému Linux vyžaduje zmenu alebo doplnenie jeho bezpečnostných mechanizmov. Na to je vhodné využiť nejaké už existujúce riešenie, ktoré poskytuje vhodné rozhranie na uplatňovanie inej bezpečnostnej politiky. Vhodným riešením je projekt Medusa DS9, ktorý za pomoci virtuálnych svetov efektívne uplatňuje politiku definovanú autorizačným serverom. Autor tejto práce je navyše aj spoluautorom projektu Medusa DS9.

Pre systematické začlenenie rôznych bezpečnostných politík do operačného systému Linux je potrebné definovať jednotnú abstrakciu všetkých objektov a práv v systéme. To bude umožňovať aby mohli tieto rôzne bezpečnostné politiky spolupracovať a hlavne aby mohli byť uplatňované jedným spoločným mechanizmom. V tomto prípade VS monitorom v spolupráci s autorizačným serverom.

Z uvedeného vyplýva, že s použitím projektu Medusa DS9 je možné integrovať rôzne bezpečnostné politiky a rôzne bezpečnostné modely do operačného systému za pomoci autorizačného servera, ktorý pracuje mimo jadra operačného systému. Tento autorizačný server bude tvoriť rozhranie medzi VS modelom a rôznymi bezpečnostnými modelmi. Na tento účel je potrebné okrem VS modelu zaviesť aj ďalšie unifikujúce prvky. Najvýznamnejším je jednotný systém pomenovania všetkých objektov v systéme a jednotnú formu všetkých oprávnení.

4.1 Štruktúra autorizačného servera

Na to aby bol autorizačný server ľahko rozšíriteľný o ďalšie podporované bezpečnostné modely, je najvýhodnejšia jeho modulárna štruktúra (obrázok 4). Podpora jednotlivých bezpečnostných modelov je rozdelená na osobitné moduly. Jadro autorizačného servera tvorí tiež jeden samostatný modul, ktorý obsahuje komunikačné rozhranie na komunikáciu s jadrom operačného systému. Obsahuje tiež spracovanie hlavného konfiguračného súboru a poskytuje infraštruktúru pre moduly bezpečnostných modelov. Moduly využívajú len infraštruktúru poskytovanú jadrom autorizačného servera.



Obr. 4: Štruktúra autorizačného servera

4.2 Jadro autorizačného servera

Jadro autorizačného servera (infraštruktúra) poskytuje mechanizmy, ktoré budú využívať moduly rôznych bezpečnostných modelov. Moduly nekomunikujú priamo s bezpečnostným rozhraním projektu Medusa DS9 ale prostredníctvom infraštruktúry. Mechanizmy infraštruktúry teda predstavujú ďalšiu úroveň abstrakcie. Výhodou infraštruktúry je okrem abstrakcie aj fakt, že moduly môžu využívať pohodlné mechanizmy na správu niektorých dátových štruktúr, ktoré navyše takto môžu tvoriť spoločný bod všetkých bezpečnostných politík. To bude podrobnejšie objasnené ďalej.

Infraštruktúra poskytuje tieto mechanizmy, ktoré budú nižšie podrobne opísané:

1. správa VS
2. jednotný priestor mien
3. abstrakcia typov prístupov
4. správa cieľov (target)
5. práca s komunikačnými objektami
6. správa subjektov
7. distribúcia obsluhy udalostí
8. preklad a interpretácia konfiguračného súboru

4.2.1 Správa VS

V systéme Medusa DS9 sa s množinami VS modelu pracuje ako s bitovými poliami. Každý bit pritom zodpovedá jednému virtuálnemu svetu. Správa VS umožňuje virtuálne svety pomenovať. Vytvára abstrakciu alokovania a potom používania virtuálneho sveta. Pri alokácii dôjde k rezervovaniu jedného bitu bitového pola VS a jeho pomenovaniu.

4.2.2 Jednotný priestor mien

Jednoznačná identifikácia všetkých objektov v systéme je zabezpečená už samotným operačným systémom. V operačných systémoch typu UNIX je žiaľ identifikácia objektov rôzneho typu rôzna. Súbory sú napríklad jednoznačne identifikované číslom i-uzlu (inode) a číslom blokového zariadenia, na ktorom sú uložené. Používateľským programom však takáto identifikácia súborov nie je umožnená. Pre ne je použiteľná identifikácia absolútnou cestou, teda podľa mena súboru a plnej cesty z koreňového adresára. Takáto identifikácia už nie je úplne jednoznačná, lebo je možné celé disky pripájať na rôzne cesty a čo je dôležitejšie, UNIX umožňuje vytvárať viacnásobné odkazy na súbory, takzvané hardlinky, čo je prakticky vytvorenie viacerých ciest k tomu istému súboru. Teda jednej ceste môže zodpovedať viacero súborov (pripájanie diskov) a jeden súbor môže mať viacero ciest (hardlinky).

Identifikácia na základe cesty v súborovom strome je pre používateľa ľahko predstaviteľná a pritom dostatočne presná. Dôležitým aspektom stromovej štruktúry je skutočnosť, že súbory v nej uložené môžu a prakticky aj sú logicky zoradené tak, že súbory podobného významu sa nachádzajú v jednej alebo len v malom množstve vetiev. Ak existuje spôsob ako naraz identifikovať všetky súbory v nejakom podadresári, tak je potom jednoduché identifikovať všetky súbory podobného významu.

Systém UNIX vychádza z idei, že objekty v systéme sú súbory. Ak by bola táto idea dôsledne dodržaná, existoval by jednotný systém pomenovania všetkých objektov - jednotný priestor mien. Skutočnosť je však iná. Príkladom sú procesy, ktoré sú identifikované svojim identifikačným číslom PID. Podobne sú na tom objekty medziprocesovej komunikácie ako aj všetky sieťové objekty (sockety, spojenia ...).

Operačný systém potrebuje rozlišovať medzi rôznymi typmi objektov len z funkčného hľadiska. Bezpečnostné politiky nepotrebujú typy objektov rozlišovať. Stačí im objekt identifikovať, aby bolo možné určiť prístupové práva. Preto je vhodné ak je možné všetky objekty identifikovať jednotným spôsobom nezávisle na ich type. Na to slúži zavedený jednotný priestor mien.

Navrhovaný systém pomenúvania objektov vychádza zo stromovej štruktúry súborov. Ten je rozšírený o jeden hlavný uzol zaradený nad koreňový uzol súborového systému. Z hlavného uzla budú okrem vetvy súborového systému vychádzať aj ďalšie vetvy pre rôzne typy objektov. Nemusí sa pritom jednať len o systémové objekty. Môže ísť aj o rôzne objekty definované rôznymi bezpečnostnými modelmi.

Strom jednotného priestoru mien slúži len na pomenovanie objektov, nie na udržiavanie skutočného stavu systému, aj keď skutočný stav do určitej miery odzrkadľuje. Strom jednotného priestoru mien umožňuje pomenovať možné objekty bez ohľadu na to, či existujú alebo nie.

Strom jednotného priestoru mien je jedným z najvýznamnejších miest, kde sa stretávajú všetky moduly autorizačného servera. Moduly môžu definovať nové vetvy pod hlavným koreňovým uzlom a súčasne môžu identifikovať objekty zaradené na ľubovoľnom mieste v strome. Táto skutočnosť je jednou z hlavných záruk spolupráce rôznych bezpečnostných politík a modelov.

4.2.3 Abstrakcia typov prístupov

Ak vychádzame z predstavy, že každá operácia v systéme sa dá popísať trojicou (operácia, subjekt, objekt), tak aj právo vykonať túto operáciu je vlastne taká istá trojica. Nositeľmi práv sú subjekty. Preto stačí ak má každý subjekt definované svoje práva ako dvojice (operácia, objekt). V súlade s VS modelom a množinami $\mathbf{SVS_a(s)}$ môžeme operácie rozdeliť na niekoľko typov a objektom určiť ich príslušnosť $\mathbf{OVS(o)}$. Takýmto spôsobom je možné ľubovoľné práva nad ľubovoľným objektom transformovať do VS modelu.

Medusa DS9 rozlišuje v jadre operačného systému tri typy prístupu (read, write, see). Toto môže byť postačujúce na riadenie prístupu k systémovým objektom UNIXu. Na riadenie prístupu k objektom rôznych bezpečnostných modelov však môže byť potrebné rozlišovať viacero typov prístupov.

Infraštruktúra autorizačného servera rozlišuje sedem typov prístupov:

READ - Čítanie informácií z objektu

WRITE - Zápis informácií do objektu

SEE - Zisťovanie existencie objektu

CREATE - Vytváranie nového objektu (nie systémového)

ERASE - Rušenie objektu (nie systémového)

ENTER - Nadobúdanie stavu. (Zmena subjektu, ktorá spôsobí zmenu jeho zaradenia v strome. napr. exec())

CONTROL - Modifikovanie vlastností objektu.

Každý subjekt má teda priradených sedem VS množín svojich schopností. V jadre operačného systému sa síce uplatnia len prvé tri ale autorizačný server môže robiť rozhodnutia na základe všetkých siedmich.

4.2.4 Správa cieľov (target)

Strom je prehľadný nástroj na pomenovanie objektov, ale objektov existuje v systéme veľa a v praxi je často potrebné dávať právo prístupu k celej skupine objektov. Takúto skupinu často nemá zmysel deliť na jednotlivé objekty. Príkladom takejto skupiny môžu byť všetky systémové programy. Z hľadiska VS modelu to znamená. že jedna skupina predstavuje jeden virtuálny svet a teda všetky objekty, ktoré do nej patria, majú tento virtuálny svet v množine $\mathbf{OVS}(\mathbf{o})$. Skupina objektov bola nazvaná cieľ (target) preto, že slúži na prehľadnejšie definovanie práv. Právo je potom dvojica (typ prístupu, cieľ).

Infraštruktúra poskytuje správu cieľov. Umožňuje vytvárať pomenované ciele a priraďovať im ľubovoľné množstvo objektov. Pritom je možné cieľu priradiť buď konkrétny objekt alebo všetky objekty patriace nejakej vetve stromu objektov.

Cieľu je možné priraďovať práva. To znamená priradiť práva všetkým objektom v cieli, ktoré môžu v systéme vystupovať ako subjekty (v operačnom systéme Linux objektom, ktoré reprezentujú procesy). Je teda možné napríklad dať cieľu *p1* právo čítať cieľ *s1*.

Cieľ je teda skupina objektov alebo subjektov zo stromu jednotného priestoru mien. Keďže aj subjekty niekedy vystupujú ako objekty operácií, je možné jednoducho povedať, že cieľ je skupina objektov zo stromu jednotného priestoru mien.

4.2.5 Práca s komunikačnými objektami

Komunikačným rozhraním s jadrom operačného systému sú prenášané informácie o objekte a subjekte operácie. Štruktúra týchto informácií je daná jadrom operačného systému. Projekt Medusa DS9 presne definuje štruktúru prenášaných dát. Projekt Medusa DS9 si v súčasnosti kladie za cieľ zmenu protokolu existujúceho komunikačného rozhrania. Nový komunikačný protokol nebude presne definovať štruktúru prenášaných informácií. Štruktúru týchto informácií však bude umožňovať zistiť. Bude popísaná mennými názvami atribútov, ich typom, veľkosťou a umiestnením v bloku dát.

Autorizačný server je vnútorne navrhovaný na prácu s novým komunikačným protokolom. K informáciám o objektoch získaným od jadra nepristupuje pomocou pevne definovanej štruktúry ale pomocou menného popisu ich štruktúry.

Infraštruktúra autorizačného servera tento mechanizmus poskytuje modulom aj na ich vlastné použitie, napríklad ako štruktúrované premenné v konfiguračnom jazyku.

4.2.6 Správa subjektov

Infraštruktúra obsahuje mechanizmus na zistenie $\mathbf{SVS_a(s)}$ subjektu $\boldsymbol{s}$ pre typ prístupu $\boldsymbol{a}$. Rôzne moduly môžu dávať subjektu rôzne práva na základe rôznej logiky, preto musia byť všetky vyzvané aby nastavili práva subjektu. Výsledné práva sú zjednotením práv od všetkých modulov.

4.2.7 Distribúcia obsluhy udalostí

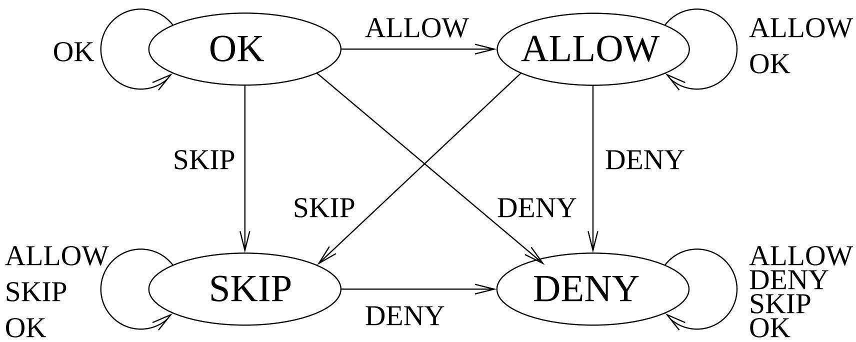
Každá žiadosť jadra operačného systému o schválenie operácie alebo inicializáciu bezpečnostných informácií objektu je považovaná za udalosť (event).

Udalosť nesie informácie o subjekte a objekte operácie a tiež informácie o samotnej operácii.

Infraštruktúra musí zabezpečiť, že sa každá udalosť dostane na spracovanie do každého modulu, ktorého sa týka. To je zabezpečené tým, že sa moduly zaregistrujú na obsluhu udalosti.

Pri výskyte udalosti sú zavolané všetky zaregistrované obslužné funkcie. Každá z nich môže modifikovať informácie o subjekte (jeho práva, prípadne dalšie prídavné informácie) a každá z nich vyhlási jednu zo štyroch možných odpovedí.

Výsledná odpoveď autorizačného servera je potom najpriamejšia a najreštriktívnejšia odpoveď zo všetkých dielčich odpovedí. Výsledná odpoveď zodpovedá konečnému stavu automatu popísaného stavovým diagramom na obrázku 5.



Obr. 5: Diagram vyhodnotenia rozhodnutí

4.2.8 Preklad a interpretácia konfiguračného súboru

Súčasťou infraštruktúry je aj preklad a interpretácia konfiguračného súboru. V konfiguračnom súbore je možné definovať ciele (target), priraďovať im práva a definovať rozhodovacie pravidlá pre operácie.

Konfiguračný súbor ale nie je jedinou možnosťou konfigurácie. Jednotlivé moduly môžu na základe vlastných konfiguračných súborov a na základe vlastnej politiky definovať všetko to, čo je možné definovať aj v konfiguračnom súbore.

Cieľ (target) sa definuje vymenovaním všetkych ciest k objektom alebo vetvám stromu. V konfiguračnom súbore sa vetvy od objektov odlišujú kľúčovým slovom *recursive*. Práva sa prideľujú cieľom, prakticky sa priradia všetkým subjektom, ktoré zahŕňa daný cieľ. Právo je dvojica (typ práva, cieľ). V konfiguračnom súbore sa teda jednému cieľu dáva právo na iný cieľ. Podrobnejšie je formát konfiguračného súboru opísaný v používateľskej príručke.

Príklad:

```
target procesy recursive "proc"; /* vsetky procesy */
target moje_subory recursive "fs/home/marek/moje";
procesy READ moje_subory
```

Definovanie rozhodovacích pravidiel je ďalšou dôležitou možnosťou konfiguračného súboru. V konfiguračnom súbore je možné definovať rozhodovacie pravidlo, ktoré sa uplatní ak subjekt z jedného cieľa robí operáciu nad objektom z druhého cieľa. Rozhodovacie pravidlo je vlastne obsluha udalostí zapísaná programom, ktorý sa vykoná počas rozhodovania o operácii. Programovací jazyk rozhodovacieho pravidla je veľmi podobný jazyku C. Obsahuje plnohodnotné vyhodnocovanie výrazov so všetkymi operátormi jazyka C okrem pretypovaní a práce s ukazovateľmi. Zachováva priority ako sú definované v jazyku C. Jazyk tiež obsahuje všetky riadiace konštrukcie jazyka C, okrem príkazu goto. V programe je možné pracovať s informáciami o objekte a subjekte operácie ako aj s atribútmi operácie ako so štruktúrami pomocou operátora bodka '.'. Pritom je umožnená aj skrátená forma prístupu na tieto atribúty. Ak to nie je nevyhnutné pre jednoznačnosť, je možné pristupovať priamo na atribúty operácie, subjektu a objektu len pomocou ich mena. Subjekt, objekt a operácia vystupujú v rozhodovaciom pravidle pod mennými názvami, ktoré sú definované pomocou komunikačného protokolu jadrom operačného systému, napríklad pri operácii *exec* subjekt vystupuje pod menom *proc* a objekt pod menom *file*. Výsledné rozhodnutie o povolení alebo nepovolení operácie sa robí pomocou príkazu *return*.

Príklad:

```
procesy exec moje_subory {
        if ( proc.uid != file.uid && proc.uid!=0 )
                return DENY;
        return ALLOW;
}
```

Program rozhodovacieho pravidla je pri inicializácii autorizačného servera preložený do vnútornej formy blízkej strojovému kódu. Jeho vykonávanie je tak časovo menej náročné a taktiež sa syntaktické chyby odhalia v čase prekladu.

Pre zjednodušenie je na miestach, kde má byť v konfiguračnom súbore uvedený cieľ, možné priamo uviesť v uvodzovkách cestu. Toto je považované za vytvorenie a použitie nepomenovaného cieľa, ktorý obsahuje len zadanú cestu.

4.3 Moduly autorizačného servera

4.3.1 Modul základného bezpečnostného modelu

Súčasťou základného bezpečnostného modelu je zaradenie súborov a procesov do stromu jednotného priestoru mien. Zaradenie súborov je jednoduché. Pod hlavným uzlom sa vytvorí uzol **fs**, ktorý predstavuje koreňový adresár súborového systému. Pod ním sa nachádzajú súbory zoradené tak ako v súborovom systéme. Problémom sú hardlinky, lebo tie umožňujú aby bol jeden súbor (jeden i-uzol) súčasne na viacerých miestach v strome. Pritom všetky výskyty sú rovnocenné, žiaden nie je primárny. Hardlinky predstavujú nevyriešený problém už v samotnej Meduse. Jednoduchým riešením problémov s nimi by bolo nepoužívať ich a zakázať ich používanie. V praxi sa v systémoch typu UNIX používajú len zriedka. Výskyt hardlinkov v rámci jedného adresára prevažne nebýva problém, lebo často všetky súbory v jednom adresári patria do tých istých cieľov. Vhodným rozhodovacím pravidlom je možné povoliť vytváranie len takých hardlinkov, ktoré nebudú predstavovať bezpečnostný problém.

Procesy sa hierarchicky zaraďujú pod uzol **proc**, ktorý sa nachádza pod hlavným uzlom. Hierarchickú štruktúru vytvárajú na základe spúšťania súborov. Keď proces vykoná operáciu *exec* na súbor, tak sa automaticky posunie v strome na poduzol zodpovedajúci cieľu, v ktorom sa vykonávaný súbor nachádza. Keďže sa súbor môže naraz nachádzať vo viacerých cieľoch, bol zavedený primárny cieľ (primary target). Každý objekt sa môže naraz nachádzať len v jednom primárnom cieli. Hierarchia v strome procesov je založená práve na primárnych cieľoch vykonávaných súborov.

Príklad:

```
primary target bash "fs/bin/bash";
"proc/bash" READ moje_subory;
```

V strome procesov sa navyše uplatňuje aj právo ENTER. Procesu je povolené spustiť len také súbory, ktoré ho presunú do uzlov, na ktoré má právo ENTER.

4.3.2 Modul Doménového modelu

Usporiadanie procesov do stromovej štruktúry nemusí byť vždy vhodné. Ďalšou alternatívou je rozdelenie procesov do domén. Každý proces patrí práve do jednej domény. Práva potom môžeme priraďovať týmto doménam. Taktiež je možné stanoviť rozhodovacie pravidlá na operácie mezi doménou a inými objektami. Takéto pravidlá sa potom týkajú všetkých procesov v tejto doméne. Zmena domény procesu sa môže realizovať v rozhodovacom pravidle pomocou príkazu *domain*.

Doménový model definuje uzol **domain** pod hlavným koreňovým uzlom. Pod uzlom domain sa nachádzajú všetky domény bez ďalšieho hierarchického usporiadania.

Takýchto modelov je možné vytvoriť ľubovoľné množstvo. Ich účelom je poskytnúť iný pohľad na usporiadanie procesov, ktorý je v niektorých prípadoch vhodnejší ako iné.

4.3.3 Modul RBAC modelu

Bezpečnostný model RBAC začleňuje do autorizačného servera samostatný modul. Tento modul zabezpečuje uplatňovanie politiky bezpečnostného modelu RBAC. Modul tiež zaraďuje niektoré komponenty RBAC modelu do stromu jednotného priestoru mien, aby nad nimi mohli byť definované práva. Modul RBAC pracuje na základe vlastného konfiguračného súboru.

Modul RBAC definuje pod hlavným koreňovým uzlom uzol **rbac**, pod ktorým sa nachádza niekoľko vetiev zodpovedajúcich rôznym častiam RBAC modelu.

Ďalej nasleduje návrh začlenenia jednotlivých komponentov RBAC modelu do autorizačného servera.

4.3.3.1 Množina používateľov

Množina používateľov je inicializovaná podľa štandardného UNIXového súboru /etc/passwd. V ňom sú uložené mená a identifikačné čísla (UID) používateľov.

Používatelia sú zaradení v strome pod uzlom **rbac/user**. Napríklad cesta k používateľovi **root** je **„rbac/user/root”**.

4.3.3.2 Množina rolí

Role sú je definované v konfiguračnom súbore RBAC modulu. Každá rola je identifikovaná svojim jedinečným menom. Do stromu jednotného priestoru mien sú role zaradené pod uzol **rbac/role**. Priamo pod týmto uzlom sa nachádzajú všetky role. Role však majú svoju hierarchiu ako to požaduje $RBAC_1$.

Táto hierarchia je v strome vystihnutá tým, že ak do cieľa zaradíme namiesto konkrétnej role celý podstrom pod touto rolou (kľúčové slovo recursive), tak to znamená zaradenie aj všetkých podrolí danej role. Hypoteticky sú teda role pod uzlom **role** hierarchicky usporiadané.

Aby bolo možné označiť aj nadrole, je zavedený uzol **ROLE** pod uzlom **rbac**, ktorý má taký istý význam ako uzol **role**, ale pod ním sú role hypoteticky usporiadané v opačnej hierarchii.

Príklad:

`"rbac/role/technik"`	rola technik
`"rbac/ROLE/technik"`	rola technik
`recursive "rbac/role/technik"`	rola technik a jej podrole
`recursive "rbac/ROLE/technik"`	rola technik a jej nadrole

4.3.3.3 Množina práv

Právo je určené dvojicou (typ prístupu, cieľ). Množina všetkých práv je množina všetkých možných takýchto dvojíc. Aby boli aj práva postihnuteľné ako objekty systému, sú zaradené do stromu pod uzol **rbac/permission**. Pod týmto uzlom sú ciele práv a pod každým cieľom sú typy prístupu. Takéto poradie umožní jednoduchšie určiť všetky práva na jeden cieľ než jeden typ práva na všetky ciele. To viac zodpovedá praktickým požiadavkám.

Príklad pomenovania práva čítať target *moje_subory*:

```
"rbac/permission/moje_subory/READ"
```

Príklad pomenovania všetkých práv na target *moje_subory*:

```
recursive "rbac/permission/moje_subory"
```

4.3.3.4 Relácia priradenia používateľov

Priradenie používateľov rolám je definované v konfiguračnom súbore RBAC modulu. Je udržiavané modulom RBAC. Nie je potrebné aby bolo nejakým spôsobom zaradená do stromu, lebo nepredstavuje objekt nad ktorým sú vykonávané operácie. Operácie sú vykonávané nad rolami a používateľmi.

4.3.3.5 Relácia priradenia práv

Priradenie práv rolám je taktiež definované v konfiguračnom súbore RBAC modulu. RBAC modul ho pri inicializácii prevedie na formu zodpovedajúcu VS modelu. Každá rola **r** má teda priradené množiny $\mathbf{SVS_a(r)}$, ktoré sa pridajú k monžinám $\mathbf{SVS_a(s)}$ subjektu **s** pri nadobudnutí role **r**.

4.3.3.6 Relácia prihlásenie

Relácia prihlásenie sa vytvára dynamicky počas behu systému. Je udržiavaná modulom RBAC, ktorý musí sledovať vznik a zánik každého prihlásenia. Aby bolo možné zrušiť prihlásenia, do ktorých už nepatrí žiaden proces, je nutné sledovať všetky vzniky a zániky procesov v systéme.

4.3.3.7 Hierarchia rolí

Hierarchia rolí je definovaná konfiguračným súborom RBAC modulu. V strome je hierarchia rolí zakomponovaná vo vetvách **rbac/role** a **rbac/ROLE**, ako bolo popísané vyššie.

4.3.3.8 Množina obmedzení

Množina obmedzení by mala byť definovaná v osobitnom konfiguračnom súbore RBAC modulu. Obsahuje predikáty, ktoré je možné pri inicializácii RBAC modulu preložiť do vnútornej formy. Pri každej zmene v RBAC modele je potrebné pred samotnou zmenou overiť, či zmena neodporuje obmedzeniam. Toto je vhodné vykonať interpretáciou preloženej vnútornej formy obmedzení, tým sa dosiahne efektívne uplatňovanie obmedzení. V predikátoch by mali byť dostupné rôzne komponenty RBAC modelu ako objektové premenné.

4.3.3.9 Správa a definovanie RBAC politiky

Modul RBAC autorizačného servera umožňuje aj správu RBAC počas behu systému. Pritom sa uplatňujú práva vzťahujúce sa na samotné RBAC.

Rola, ako každý objekt v systéme, má množinu **OVS(o)**, ktorá určuje jej príslušnosť k virtuálnym svetom. Subjekt, ktorý má na nejakú rolu právo CONTROL (dalo by sa povedať, že je administrátorom tejto role) je oprávnený priraďovať (pridávať a uberať) tejto roli práva a používateľov.

Používateľ ako objekt systému má tiež svoju množinu **OVS(o)**, ktorá určuje jeho príslušnosť k virtuálnym svetom. Subjekt, ktorý má na nejakého používateľa právo CONTROL (dalo by sa povedať, že je jeho administrátorom), môže tohto používateľa priraďovať rolám. Inač povedané môže mu pridávať a uberať role.

Na to, aby mohol subjekt priradiť nejakého používateľa nejakej role, musí mať právo CONTROL aj na používateľa aj na rolu.

Právo ako objekt systému má taktiež svoju množinu **OVS(o)**, ktorá určuje jeho príslušnosť k virtuálnym svetom. Subjekt, ktorý má na nejaké právo právo CONTROL, je oprávnený toto právo priraďovať (pridávať a uberať) rolám.

Na to aby mohol subjekt priradiť nejaké právo nejakej role musí mať právo typu CONTROL aj na toto právo aj na rolu.

Takto sa vždy uplatňujú práva na oba objekty operácie. To znižuje počet potrebných typov prístupov na jeden typ a tým je typ CONTROL. Pritom sa vždy zachováva filozofia práva ako dvojice (typ prístupu, cieľ).

Právo vytvárať nové role predstavuje právo CREATE na objekt **rbac/role** alebo **rbac/ROLE**. Právo rušiť role predstavuje právo ERASE na danú rolu alebo na objekt **rbac/role** alebo **rbac/ROLE**.

Právo modifikovať hierarchiu rolí vyžaduje právo CREATE na obe role, ktorým je určovaný ich vzájomný vzťah. Vychádza to z predstavy, že role sú v strome usporiadané hierarchicky a tak vytvoriť podrolu alebo nadrolu je vlastne operácia vytvárania, čiže CREATE.

Správu RBAC zabezpečuje samotný modul RBAC. Na komunikáciu používateľa s týmto modulom slúži externý program **rbacadm**, ktorý len odovzdá svoje argumenty modulu RBAC. Viac o tomto programe bude popísané v kapitolách implementácia a používateľská príručka.

5 Implementácia

Autorizačný server je implementovaný v jazyku C tak, aby v čo najväčšej miere zodpovedal špecifikácii a návrhu. Veľký dôraz bol kladený na efektívnosť. Podrobným popisom implementovaného autorizačného servera sa zaoberá kapitola Technická dokumentácia. Hlavičkové súbory zdrojového textu autorizačného servera sú v prílohe.

Program **rbacadbm**, slúžiaci na administráciu RBAC modelu, je taktiež implementovaný v jazyku C. Obsahuje len jedno volanie jadra operačného systému - syscall(), ktoré pomocou jadra odovzdá parametre programu **rbacadm** autorizačnému serveru. Ten ich potom vyhodnotí a vykoná požadovanú operáciu. Komunikácia pomocou systémového volania bola zvolená preto, lebo pre autorizačný server je najvýhodnejšie ak požiadavku na vykonanie operácie dostane od jadra operačného systému aj spolu s právami a atribútmi procesu, ktorý operáciu požadoval. Aby sa toto udialo, program **rbacadm** musí požadovať operáciu, ktorá spôsobí, že sa na povolenie tejto operácie jadro obráti na autorizačný server. Pritom je potrebné preniesť niekoľko argumentov zo žiadajúceho procesu do RBAC modulu autorizačného servera. Na tento účel je najvhodnejšie použiť v Linuxe nevyužité systémové volanie.

6 Výsledky riešenia a ich zhodnotenie

Výsledkom tejto práce je modulárny autorizačný server s infraštruktúrou, ktorá umožňuje čo najpohodlnejšie a najefektívnejšie začlenenie rôznych bezpečnostných politík a modelov do operačného systému Linux.

Autorizačný server obsahuje základné moduly pre prácu so základnými systémovými objektami (súbory, procesy). Možnosti autorizačného servera sú limitované možnosťami projektu Medusa DS9. Projekt Medusa DS9 uplatňuje VS model aj nad objektami IPC ale iba tak, že IPC objekt dedí VS množinu svojej príslušnosti $\mathbf{OVS}(\mathbf{o})$ od procesu, ktorý ho vytvoril. Ak by jadro operačného systému pomocou komunikačného protokolu žiadalo od autorizačného servera inicializáciu bezpečnostných informácii IPC objektu, alebo by žiadalo o schválenie operácií nad IPC objektami, potom by bolo jednoducho možné do autorizačného servera pridať aj podporu IPC objektov a zaradiť ich do stromu jednotného priestoru mien. Podobne je to realizovateľné aj s ostatnými systémovými objektami. Je reálne predpokladať, že sa vývoj projektu Medusa DS9 bude uberať aj týmto smerom.

Projekt Medusa DS9 zatiaľ neuplatňuje VS model nad objektami siete. Podobne nie je možné sledovať operácie vykonávané nad sieťovými objektami. Navrhnutý a implementovaný autorizačný server v tomto nepredbieha projekt Medusa DS9. Nie je totiž ešte isté, ako budú v Meduse DS9 sieťové objekty reprezentované. O neskoršej implementácii sieťových objektov platí, obdobne ako o IPC, že bude jednoduché pridať podporu týchto objektov.

Bezpečnostný model RBAC je implementovaný ako modul autorizačného servera. Modul RBAC umožnuje aj administráciu RBAC modelu počas behu systému, pričom zabezpečuje riadenie prístupu na objekty RBAC modelu. Na tento účel definuje administratívne práva rovnakým spôsobom ako sú definované aj všetky ostatné práva v systéme, teda pomocou abstrakcie typov prístupov a pomocou jednotného priestoru mien, do ktorého začleňuje aj objekty RBAC modelu. Z modelu RBAC nie je implementovaná množina obmedzení a relácia prihlásenie.

Podobne ako RBAC môžu byť do autorizačného servera, a tak aj do operačného systému Linux, začlenené aj dalšie bezpečnostné modely. Na to, aby ich bolo možné implementovať, je potrebné nájsť ich vhodné namapovanie na VS model definovaný projektom Medusa DS9.

6.1 Obmedzenia riešenia

Obmedzenia tohto riešenia sú do značnej miery totožné s obmedzeniami projektu Medusa DS9. Okrem nich implementovaný autorizačný server prináša niekoľko obmedzení, ktoré sú však relatívne jednoducho zmeniteľné. Ide o limit maximálneho počtu rozlišiteľných operácií nad objektami jedného typu, ktorý predstavuje maximálne 32 operácii na jeden typ objektu. To je viac ako používa súčasná verzia projektu Medusa DS9. Dostupná operačná pamäť a veľkosť adresného priestoru tiež obmedzujú autorizačný server, lebo obmedzujú kapacitu jeho tabuliek a štruktúr.

Dalším obmedzením je skutočnosť, že jedna časť autorizačného servera, zaoberajúca sa interpretáciou rozhodovacieho pravidla, je závislá od poradia bajtov v slove, ktoré používa procesor. Implementácia momentálne podporuje len poradie bajtov *little endian*.

Maximálna veľkosť celočíselného atribútu objektu je limitovaná na 64 bitov. Veľkosť a offset bitových polí musí byť zarovnaný na bajty.

6.2 Perspektívy riešenia

Projekt Medusa DS9 má v budúcnosti prejsť na nový komunikačný protokol, ktorý bude viac objektovo orientovaný. Pritom bude realizovaná podpora ďalších systémových objektov. Doplnenie implementácie podpory v autorizačnom serveri bude jednoduchá.

Projekt Medusa DS9 má ambíciu pracovať nielen s operačným systémom Linux ale aj s ďalšími operačnými systémami typu UNIX. Ak bude existovať implementácia projektu Medusa DS9 aj pre iný operačný systém, potom bude použiteľný aj implementovaný autorizačný server, ktorý nie je závislý od operačného systému.

7 Overenie výsledkov a testovanie

Autorizačný server je pomerne zložitý program, takže nie je jednoduché overiť jeho funkčnosť. V priebehu procesu implementácie bola testovaná funkčnosť jeho jednotlivých častí. Testovanie malo za cieľ overiť funkčnosť všetkých častí ako aj overiť funkčnosť autorizačného servera ako celku. Výsledok testovania bol pozitívny. Takéto testovanie však nedokáže odhaliť zložitejšie chyby. Na kvalitné otestovanie autorizačného servera je nutné nasadiť ho v praxi a to na niekoľko vyťažených systémov. Praktické použitie najlepšie ukáže aj skryté chyby.

Aj z toho dôvodu bude autorizačný server zverejnený prostredníctvom Internetu a tak daný k dispozícii každému, kto oň bude mať záujem. Takto bude distribuovane testovaný.

Ďalším problémom je overenie, nakoľko sú mechanizmy poskytované autorizačným serverom postačujúce na uspokojenie požiadaviek používateľov. Toto je možné overiť jedine v praxi. Je pravdepodobné, že prax si vyžiada doplnenie autorizačného servera o ďalšie vlastnosti a mechanizmy.

8 Technická dokumentácia

Autorizačný server je naprogramovaný v jazyku C. Zdrojové texty sú rozdelené tak, aby boli čo najprehľadnejšie. Jednotlivé moduly rôznych bezpečnostných modelov sú umiestnené v podadresároch. V osobitnom podadresári je aj komunikačné rozhranie s jadrom operačného systému, spracovanie a interpretácia konfiguračného súboru.

Popis adresárov:

basic - modul základného bezpečnostného modelu

domain - modul doménového modelu

rbac - modul RBAC modelu

comm0 - kominukačné rozhranie s Medusou DS9

comm - nové komunikačné rozhranie

language - preklad a interpretácia konfiguračného jazyka

8.1 Infraštruktúra autorizačného servera

8.1.1 Správa VS

Správa VS sa nachádza v súboroch **vs.c** a **vs.h**. Pri inicializácii je potrebné stanoviť skutočný počet virtuálnych svetov pomocou funkcie *vs_init(int max_vs)*. Skutočný počet musí zodpovedať počtu jadrom operačného systému podporovaných virtuálnych svetov. Tento počet je potrebné zistiť pomocou komunikačného protokolu.

Alokácia virtuálneho sveta sa uskutoční pomocou funkcie *vs_ alloc(char *name)*, vyhľadanie podľa mena pomocou funkcie *vs_find(char *name)*. Každý virtuálny svet musí mať svoje jedinečné meno.

Na aritmetické operácie s množinami virtuálnych svetov slúžia funkcie *vs_clear()*, *vs_invert()*, *vs_set()*, *vs_add()*, *vs_sub()*, *vs_mask()*. Na otestovanie prieniku množín virtuálnych svetov slúži funkcia *vs_test()*.

8.1.2 Strom objektov

V infraštruktúre je definovaný dátový typ uzol stromu (*struct tree_s*). Jednotlivé moduly si môžu uzly vo svojich vetvách rozšíriť o ďalšie údaje. Každý uzol obsahuje údaje potrebné na vytvorenie stromovej štruktúry a niekoľko údajov potrebných na jednotnú správu cieľov (target) a jednotnú správu udalostí.

Každý uzol obsahuje smerník na štruktúru popisujúcu jeho typ (*struct tree_type_s*). Typ je štruktúra popisujúca veľkosť uzla daného typu v bajtoch. To je potrebné na to, aby infraštruktúra automaticky alokovala pri vytváraní uzla dosť miesta. Typ uzla obsahuje aj smerník na funkciu *add_vs*, ktorá slúži na pridanie virtuálnych svetov do niektorej z VS množín. Túto funkciu musí definovať každý typ uzla stromu. Typ tiež obsahuje smerník na funkciu *init*. Táto je automaticky zavolaná infraštruktúrou, keď vytvorí nový uzol. Slúži na inicializáciu všetkých prídavných dát v uzle stromu.

Stromová štruktúra je tvorená smerníkmi:

parent - nadradený uzol v strome

child - prvý obyčajný podradený uzol v strome

reg - prvý regexp podradený uzol v strome

next - susedný uzol - ďalší podradený uzol toho istého nadradeného uzlu

alt - alternatíva tohto uzlu v strome

Obyčajný podradený uzol má svoje jedinečné meno pod daným naduzlom. Reprezentuje jednoznačne objekt s odpovedajúcou cestou. Regexp podradený uzol má namiesto mena regulárny výraz (regexp) [REG97], ktorý umožňuje aby reprezentoval viacero poduzlov daného nadradeného uzla. Táto možnosť, okrem svojej veľkej výhody zjednodušenia pomenovania väčšieho množstva poduzlov, prináša aj nejednoznačnosť. Obyčajný uzol môže tiež patriť k uzlom, ktoré pomenúva regexp uzol. Preto bol zavedený ukazovatľ **alt**, ktorý umožňuje rýchlo nájsť aj všetky zodpovedajúce alternatívy daného uzla. S regexp uzlami je však nutné zaobchádzať v čo najmenšom rozsahu, lebo nie je možné zabezpečiť úplnú jednoznačnosť.

Na prácu so stromom objektov slúžia následovné funkcie definované v súboroch **tree.c** a **tree.h**:

register_tree_type() - registrácia typu uzla stromu spojená aj s vytvorením poduzla pod hlavným koreňovým uzlom

create_path() - vytvorenie cesty v strome. Vytvorí všetky uzly v ceste

find_one() - nájde jeden poduzol pod uzlom *base*

8.1.3 Abstrakcia typov prístupov

Typy prístupov sú definované v súbore **access_types.h**. Tu je definovaný aj maximálny počet typov prístupov. Všetky typy uzlov stromu, ako aj všetky moduly autorizačného servera, musia podporovať tu definované typy prístupu.

8.1.4 Správa cieľov (target)

Správa cieľov sa nachádza v súboroch **target.c** a **target.h**. Na prácu s cieľmi slúži šesť funkcií:

target_create() - vytvorenie nového cieľa

target_find() - vyhľadanie cieľa podľa mena

target_add_path() - pridanie cesty v strome k cieľu *target_add_event()* - zapnutie sledovania stanovenej operácie medzi stanovenými cieľmi

target_add_vs() - pridanie virtuálnych svetov do jednej z VS množín všetkých objektov cieľa

target_get_vs() - získanie virtuálneho sveta daného cieľa

8.1.5 Práca s dátovými objektami

Na prácu s dátovými objektami slúžia štruktúry a funkcie v súboroch **object.c** a **object.h**. Štruktúry popisujúce triedy dátových objektov sú založené na štruktúrach zo súboru **medusa_object.h**, ktoré zodpovedajú novému komunikačnému protokolu medzi Medusou DS9 v jadre operačného systému a autorizačným serverom.

Štruktúra *struct class_s*, ktorá popisuje triedu objektu obsahuje aj zreťazený zoznam smerníkov na funkcie, ktoré inicializujú bezpečnostné informácie objektov tejto triedy, pre jednotlivé moduly.

Na prácu s triedami a objektami slúžia tieto funkcie:

add_class() - registrácia triedy

get_class_by_name() - vyhľadanie triedy podľa mena

get_class_by_id() - vyhľadanie triedy podľa identifikačného čísla, optimalizované na rýchlosť

get_attribute() - vyhľadanie atribútu triedy podľa mena

class_alloc_cinfo() - alokácia 32 bitového cinfo. Takto si môže každý modul autorizačného servera vyhradiť miesto na vlastné informácie, alebo smerník do vlastných štruktúr, pri objektoch nejakej triedy.

class_add_sethandler() - pridanie funkcie na inicializáciu bezpečnostných infromácii *object_is_same()* - otestovanie, či dva objekty reprezentujú ten istý objekt. Testuje sa zhodnosť primárneho kľúča oboch objektov

object_do_sethandler() - inicializácia bezpečnostných informácií objektu

8.1.6 Správa subjektov

Správa subjektov pozostáva z funkcie *subject_get_vs()* na zistenie množín $\mathbf{SVS_a(s)}$ subjektu a z funkcie *subject_register()* slúžiacej na registráciu dielčich funkcií na tento účel. Správa subjektov sa nachádza v súboroch **subject.c** a **subject.h**.

8.1.7 Distribúcia obsluhy udalostí

Distribúcia obsluhy udalostí je jednou z najzložitejších častí autorizačného servera. Nachádza sa v súboroch **event.c** a **event.h**. Pozostáva z registrácie typov udalostí, registrácie obslužných rutín pre rôzne udalosti a z funkcií na prácu s nimi. Všetky funkcie sú uvedené v prílohe.

Dôležitou funkciou je *do_event()*, ktorá vykoná celú obsluhu udalostí tak, že zistí o akú udalosť ide, aký subjekt udalosť vyvolal a akého objektu sa týka. Potom zavolá všetky obslužné rutiny, ktoré boli pre túto udalosť zaregistrované. Na tento účel sa využívajú hash tabuľky, ktoré zabezpečia čo najväčšiu rýchlosť vyhľadávania.

8.1.8 Preklad a interpretácia konfiguračného súboru

Preklad a inerpretácia konfiguračného súboru sa nachádza v osobitnom adresári **language**. Gramatika konfiguračného súboru je definovaná v niekoľkých súboroch:

expression.c - gramatika výrazov. Operátory a ich priority zodpovedajú jazyku C.

language.c - gramatika riadiacich konštrukcií if, else, do, while, for, switch, case a blok.

data.c - gramatika premenných.

conf_lang.c - gramatika a interpretácia konfigurácie cieľov a operácií.

Vykonávaním preloženého programu rozhodovacích pravidiel (obslužných rutín udalostí) sa zaoberajú súbory **execute.c**, **execute.h**, **alu.c**, **alu_little.c**. Aktuálna implementácia je funkčná len pre architektúry s *little endian* poradím bajtov. Prerobením súboru **alu_little.c** je možné zabezpečiť aj podporu iných architektúr.

8.2 Moduly autorizačného servera

Moduly autorizačného servera sú uložené v osobitných podadresároch. Každý modul má svoju inicializačnú funkciu, ktorá je určená na inicializáciu vnútorných údajov modulu a na registráciu všetkého, čo modul potrebuje na svoju korektnú činnosť.

8.2.1 Základný modul

Základný modul sa nachádza v adresári **basic**. Skladá sa z dvoch častí:

1. Funkcie zabezpečujúce zaradenie súborov do stromu jednotného priestoru mien.
2. Funkcie zabezpečujúce zaradenie procesov do stromu jednotného priestoru mien.

8.2.2 Modul domén procesov

Nachádza sa v podadresári **clasic**. Zaraďuje domény do stromu jednotného priestoru mien. Definuje funkciu na zmenu domény procesu.

8.2.3 Modul RBAC

Modul RBAC sa nachádza v adresári **rbac**. Definuje vetvu rbac stromu jednotného priestoru mien. Modul RBAC číta vlastný konfiguračný súbor v ktorom sú definované role, hierarchie rolí, priradenie používateľov a priradenie oprávnení.

Modul RBAC umožňuje aj administráciu RBAC modelu. Externý proces (napríklad spustený zo shellu) požiada, pomocou jadrom nepoužívaného systémového volania, modul RBAC autorizačného servera o nejakú zmenu RBAC modelu. Modul RBAC overí práva tohto procesu na vykonanie požadovanej zmeny. Ak sú práva dostatočné, zmenu uskutoční.

Zmenený RBAC model je potom možné uložiť naspäť do konfiguračného súboru, aby sa vykonané zmeny uplatnili aj pri dalšom štarte systému.

9 Používateľská príručka

Implementovaný autorizačný server sa v duchu terminológie projektu Medusa DS9 nazýva **Constable**. Okrem toho, že sa jedná o autorizačný server pre Medusu DS9 nemá so starým autorizačným serverom projektu Medusa DS9 takmer nič spoločné, aj keď niektoré koncepcie nového Constabla vychádzajú z koncepcií a skúseností so starým Constablom. Nový Constable by mal v praxi postupne nahradiť starého.

9.1 Popis priloženého CD

Priložené CD obsahuje:

1. túto diplomovú prácu vo formátoch PostScript, Tex, dvi a pdf
2. zdrojové texty autorizačného servera s programom rbacadm
3. zdrojové texty jadra operačného systému Linux verzia 2.2.17
4. inštalačné balíky distribúcie Linuxu SlackWare verzia 7.1

Na CD sú v príslušných adresároch uložené aj popisné súbory README.

9.2 Inštalácia

Constable vyžaduje na svoj korektný beh nainštalovaný a funkčný operačný systém Linux. V jadre operačného systému musí byť zakompilovaná Medusa DS9. Je nutné použiť upravenú verziu, ktorá má rozšírené polia cinfo. Táto verzia sa nachádza na priloženom CD. Je pravdepodobné, že ďalšie verzie Medusy DS9 budú tiež obsahovať potrebné rozšírenia.

Pri inštalácii Constabla sa postupuje následovným spôsobom:

1. tar -xzf constable.tgz
2. cd constable
3. make
4. make install

Pred skompilovaním nového Constabla príkazom *make* je možné editovať súbor *Makefile* a zmeniť adresár kam bude Constable nainštalovaný. Tiež je možné pridať parametre pre kompilátor, napríklad na optimalizáciu.

9.3 Konfiguračný súbor autorizačného servera

V konfiguračnom súbore sa definujú ciele, práva a rozhodovacie pravidlá. Nezáleží v ňom na bielych znakoch (medzery, konce riadkov ...)

9.3.1 Definícia cieľov

Ciele sú definované pomocou kľúčového slova **target**. Primárny cieľ je definovaný pomocou kľúčových slov **primary target**. Ak cestu v strome pri definícii cieľa predchádza kľúčové slovo **recursive**, tak to znamená, že do cieľa nepatrí len uzol zodpovedajúci danej ceste ale aj rekurzívne všetky jeho poduzly.

Syntax definície cieľa:

`[primary] target` *meno* `[recursive] "`*cesta*`" [,[recursive] "`*cesta*`" ... ] ;`

meno - jedinečné meno tohto cieľa.

cesta - cesta v strome jednotného priestoru mien zložená z mien uzlov na ceste od koreňa oddelených znakom '/'. Na začiatku cesty je vhodné tento znak nedávať, aby bolo možné v budúcnosti na zjednodušenie zaviesť, že cesty začínajúce znakom '/' predstavujú súbory.

9.3.2 Priradenie práv

Syntax priradenia práv:

target1 access_type target2 `[, [`*access_type*`]` *target3* ... `]` ;

target1 - cieľ, ktorému sú priraďované práva

access_type - typ prístupu (operácie), ktorý je povoľovaný. Jeden z: **READ**, **WRITE**, **SEE**, **CREATE**, **DELETE**, **ENTER**, **CONTROL**.

target2,3 - cieľ, na ktorý je povoľovaný prístup.

9.3.3 Rozhodovacie pravidlá

Syntax rozhodovacieho pravidla:

target1 op `[` *target2* `]` `{` *program* `}`

target1 - cieľ, z ktorého má byť subjekt operácie.

op - názov operácie. Názvy operácií sú definované systémom Medusa DS9.

target2 - cieľ, z ktorého má byť objekt operácie.

program - program rozhodovacieho pravidla. Syntax je takmer zhodná s jazykom C. V programe sú prístupné premenné, ktoré reprezentujú atribúty operácie, atribúty subjektu a objektu. Ich názvy sú definované systémom Medusa DS9.

9.4 Konfiguračný súbor RBAC modelu

Konfigučný súbor má následovný formát:

```
rola { "používateľ" [ , "používateľ" ...  ]  ;
       podrola [ , podrola ...  ]  ;
       typ_prístupu cieľ [ , cieľ ...  ]  ;
}
```

Každá rola má definovaných svojich používateľov, svoje podrole a svoje práva. Podrole musia byť ako role definované v súbore skôr ako sú použité ako podrole. To zároveň zabraňuje vytvoreniu cyklu v grafe hierarchie rolí.

9.5 Program rbacadm

Program **rbacadm** slúži na komunikáciu s modulom RBAC autorizačného servera. Svoje argumenty odovzdá prostredníctvom systémového volania autorizačnému serveru, ktorý rozhodne o povolení a v prípade povolenia aj uplatní požadované zmeny RBAC modelu.

Použitie programu rbacadm:

```
rbacadm <op> [<arg1> [<arg2> [<arg3>]]]
```

Argument *op* určuje typ operácie:

cr - vytvorenie novej role *arg1* (create role)

dr - zrušenie role *arg1* (delete role)

ra - určenie, že rola *arg2* je podrolou role *arg1* (role assignment)

ua - priradenie používateľa *arg2* role *arg1* (user assignment)

pa - priradenie práva prístupu *arg2* na cieľ *arg3* role *arg1* (permission assignment)

rd - zrušenie nadradenosti role *arg1* role *arg2.*

ud - zrušenie členstva používateľa *arg2* v role *arg1*.

pd - zrušenie práva prístupu *arg2* na cieľ *arg3* role *arg1*.

save - uloženie aktuálneho nastavenia do konfiguračného súboru

10 Zoznam použitej literatúry

[ARB97] SANDHU, R.S., et al.: *The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and Outline.* 2nd ACM Workshop on Role-Based Access Control, 1997.

[ARB99] SANDHU, R.S., MUNAWER, Q.: *The ARBAC99 Model for Administration of Roles.* Laboratory for Information Security Technology, 1999.

[FCK95] FERRAIOLO, D., CUGINI, J., KUHN, R.: *Role-Based access control (RBAC): Features and motivations.* Proceedings of 11th Annual Computer Security Application Conference, New Orleans, 1995

[GD72] GRAHAM, G., DENNING, P.: *Protection - principles and practice.* Spring Joint Computer Conference, AFIPS Press 1972.

[LAM71] LAMPSON, B.W.: *Protection.* 5th Princeton Symposium on Information Science and Systems, 1971.

[MACH2000] Machek, Pavel,2000. *Capabilities for elf.* http://atrey.karlin.mff.cuni.cz/~pavel/elfcap.html

[NIST93] FERRAIOLO, D.F., GILBERT, D.M., LYNCH, N.: *An examination of federal and commercial access control policy needs.* NIST-NCSC National Computer Security Conference, 1993.

[REG97] KARTTUNEN, L., et al.: *Regular Expression for Language Engineering.* Xerox Research Centre, 1997.

[RSB2000] Ott, Amon. 2000. *Rule Set Based Access Control (RSBAC) for Linux.* http://www.rsbac.de/

[SAN96] SANDHU, R.S. et al.: *Role-Based Access Control Models.* IEEE Computer, 1996

[SPE99] SPENCER, R.: The Flask Security Architecture: System Support for Diverse Security Policies. University of Utah, 1999

[ZEL2000] ZELEM, M.: *Integrácia rôznych bezpečnostných politík do OS Linux.* FEI STU Bratislava, 2000.

[ZP99] ZELEM, M., PIKULA, M.: *Zvýšenie bezpečnosti OS Linux.* FEI STU Bratislava, 1999.

[ZPO2000] ZELEM, M., PIKULA, M., OČKAJÁK, M.: *Medusa DS9.* http://medusa.fornax.sk/

Príloha A: Opis súvisu s diplomovým projektom

Táto diplomová práca je pokračovaním diplomového projektu [ZEL2000]. Diplomový projekt sa zaoberal hlavne návrhom jednotného priestoru mien a unifikáciou práv. Zaviedol sedem typov prístupu a definoval právo ako dvojicu (typ prístupu, cieľ). Zaoberal sa aj podrobnou analýzou problematiky začlenenia bezpečnostného modelu RBAC do operačného systému Linux.

Diplomový projekt navrhoval aj implementáciu začlenenia modelu RBAC do operačného systému Linux. Táto diplomová práca nadväzuje na diplomový projekt, pričom pojednáva o problematike širšie a zároveň podrobnejšie. V diplomovej práci je dôraz kladený aj na čo najefektívnejšiu a najuniverzálnejšiu implementáciu.

Príloha B: Príklad konfiguračného súboru

Konfiguračný súbor autorizačného servera

```
target subory recursive "fs";
target procesy recursive "proc";

target s1 recursive "fs/bin", recursive "fs/dev", recursive "fs/etc",
          recursive "fs/home", recursive "fs/lib", recursive "fs/proc",
          recursive "fs/sbin", recursive "fs/shlib", recursive "fs/usr",
          recursive "fs/var", "fs", recursive "fs/data";

target pokus recursive "fs/pokus";

primary target tcsh "fs/bin/tcsh";

primary target ls "fs/bin/ls";

target p_root "proc";

// procesy READ s1, WRITE s1, SEE subory, SEE procesy /*, ENTER
procesy*/;
procesy READ s1, WRITE s1, SEE subory, SEE procesy , ENTER procesy;
p_root ENTER procesy;

target sisel recursive "proc/tcsh";
sisel READ subory /*, ENTER procesy*/;

"proc/tcsh" READ subory /*, ENTER procesy*/;

procesy access "fs/medusa" {
        redirect="/usr/games/trek";
}

procesy exec "fs/usr/bin/id" {
        if( uid!=0 )
```

```
                return pid % 4;
}
```

Konfiguračný súbor modulu RBAC

```
user {  "pokus";
        READ pokus;
        SEE pokus;
}

spravca { "marek";
           user;
           WRITE pokus;
}
```

Príloha C: Hlavičkové súbory