

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-72894

**SLEDOVANIE AKTIVITY PROCESU POMOCO
CONSTABLA**

BAKALÁRSKA PRÁCA

2016

Zdenko Ladislav Nagy

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-72894

SLEDOVANIE AKTIVITY PROCESU POMOCO U
CONSTABLA

BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná informatika
Číslo študijného odboru: 2511
Názov študijného odboru: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci záverečnej práce: Mgr. Ing. Matúš Jókay, PhD.

Bratislava 2016

Zdenko Ladislav Nagy



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Zdenko Ladislav Nagy**
ID študenta: 72894
Študijný program: Aplikovaná informatika
Študijný odbor: 9.2.9. aplikovaná informatika
Vedúci práce: Mgr. Ing. Matúš Jókay, PhD.
Miesto vypracovania: Ústav informatiky a matematiky (FEI)

Názov práce: **Sledovanie aktivity procesu pomocou Constabla**

Špecifikácia zadania:

Constable je autorizačný server, ktorý rozhoduje o tom, či procesom vyvolaná aktivita je legitímna alebo nie. Na to, aby sa server dokázal automaticky učiť, aké správanie je legitímne a aké nie, je potrebné vytvoriť jednoduchý autorizačný server, ktorý bude sledovať všetky procesom vyvolané aktivity.

Úlohy:

1. Naštudujte činnosť autorizačného servera Constable.
2. Navrhnite vlastný jednoduchý autorizačný server, ktorý bude:
 - a) povoľovať každú aktivitu,
 - b) zaznamenávať prístupy k jednotlivým objektom systému.
3. Implementujte prototyp takéhoto servera.
4. Zhodnoťte prínos práce.

Zoznam odbornej literatúry:

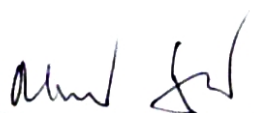
1. Zelem, M. *Integrácia rôznych bezpečnostných politik do OS LINUX*. Diplomová práca. Bratislava: FEI STU, 2001. 89 s.


Riešenie zadania práce od: 21. 09. 2015

Dátum odovzdania práce: 20. 05. 2016



Zdenko Ladislav Nagy
študent


prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska


prof. RNDr. Gabriel Juhás, PhD.
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

| | |
|---------------------------------|---|
| Študijný program: | Aplikovaná informatika |
| Autor: | Zdenko Ladislav Nagy |
| Bakalárska práca: | Sledovanie aktivity procesu pomocou Constable |
| Vedúci záverečnej práce: | Mgr. Ing. Matúš Jókay, PhD. |
| Miesto a rok predloženia práce: | Bratislava 2016 |

Práca pojednáva o bezpečnosti operačného systému Linux. Nadväzuje na prácu Ing. Jána Káčera, ktorý stojí za aktualizáciou zdrojového kódu projektu Medusa DS9 na moderné jadro Linuxu. Cieľom tejto práce je prevedenie autorizačného serveru Constable, vyvíjaného pre potreby projektu Medusa DS9, z programovacieho jazyka C do programovacieho jazyka Python. Dokumentuje celý proces preprogramovania autorizačného serveru, t.j. jeho návrh, realizáciu, testovanie a konfigurovanie potrebné pre úplnú a správnu funkčnosť autorizačného serveru. Práca taktiež popisuje zmeny, ktoré nastali pri prevode programového kódu z jazyka C do jazyka Python.

Kľúčové slová: Constable, autorizácia, Medusa Voyager, SELinux, Linux, Python, kontrola prístupu

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

| | |
|-------------------------------|---|
| Study Programme: | Applied Informatics |
| Author: | Zdenko Ladislav Nagy |
| Bachelor Thesis: | Monitoring process activity using Constable |
| Supervisor: | Mgr. Ing. Matúš Jókay, PhD. |
| Place and year of submission: | Bratislava 2016 |

This thesis deals with the security of Linux operating system. It follows the work of Ing. Ján Káčer, who stands behind porting of Medusa DS9 project source code to a modern Linux kernel. The aim of this work is to design authorization server - Constable, developed for the needs of the project Medusa DS9, from C programming language to Python. It documents the process of reprogramming authorization server, i.e. its design, implementation, testing, and configuration necessary for a full and proper functioning authorization server. The work also describes the changes that have occurred during the process of recreating the code in Python.

Keywords: Constable, authorization, Medusa Voyager, SELinux, Linux, Python, access control

Podakovanie

Chcem sa podakovať vedúcemu záverečnej práce, ktorým bol Mgr. Ing. Matúš Jókay, PhD., za odborné vedenie, rady a pripomienky, ktoré mi pomohli pri vypracovaní tejto bakalárskej práce.

Obsah

| | |
|--|-----------|
| Úvod | 1 |
| 1 Analýza problematiky | 2 |
| 1.1 Bezpečnosť | 2 |
| 1.2 Bezpečnostná politika a bezpečnostný model | 3 |
| 1.3 Bezpečnosť v OS Linux | 3 |
| 1.3.1 Riadenie prístupu k súborom | 4 |
| 1.4 Doplnkové riešenia zabezpečenia | 4 |
| 1.5 Riešenia založené na LSM | 5 |
| 1.5.1 SELinux | 5 |
| 1.5.2 AppArmor | 6 |
| 1.5.3 Smack | 6 |
| 1.5.4 TOMOYO | 7 |
| 1.6 Riešenia bez použitia LSM | 8 |
| 1.6.1 grsecurity | 8 |
| 2 Medusa Voyager | 9 |
| 2.1 Komunikačný protokol | 9 |
| 2.1.1 Pozdrav | 10 |
| 2.1.2 Inicializačné správy | 12 |
| 2.1.3 Autorizačné správy | 12 |
| 2.1.4 Režijné správy | 14 |
| 3 ReStable | 16 |
| 3.1 Špecifikácia požiadaviek | 16 |
| 3.2 Návrh | 17 |
| 3.2.1 Modul loader | 18 |
| 3.2.2 Modul event | 18 |
| 3.2.3 Modul msg_handler | 18 |
| 3.2.4 Modul logger | 18 |
| 3.2.5 Modul constants | 19 |
| 3.3 Implementácia riešenia | 19 |
| 3.4 Overenie funkčnosti a problémy | 19 |
| Záver | 21 |

| | |
|-----------------------------------|----|
| Zoznam použitej literatúry | 22 |
| Prílohy | I |
| A Štruktúra elektronického nosiča | II |

Zoznam obrázkov a tabuliek

| | | |
|------------|---|----|
| Obrázok 1 | Komunikačný diagram | 11 |
| Tabuľka 1 | Formát správy „ <i>pozdrav</i> “ | 12 |
| Tabuľka 2 | Formát správy „ <i>podporovaný k-objekt</i> “ | 12 |
| Tabuľka 3 | Formát správy „ <i>podporovaná udalosť</i> “ | 13 |
| Tabuľka 4 | Štruktúra dátového typu „ <i>attribute</i> “ | 13 |
| Tabuľka 5 | Formát správy „ <i>autorizačná žiadosť</i> “ | 14 |
| Tabuľka 6 | Formát správy „ <i>autorizačná odpoveď</i> “ | 14 |
| Tabuľka 7 | Formát správy „ <i>fetch žiadosť</i> “ | 14 |
| Tabuľka 8 | Formát správy „ <i>fetch odpoveď</i> “ | 15 |
| Tabuľka 9 | Formát správy „ <i>update žiadosť</i> “ | 15 |
| Tabuľka 10 | Formát správy „ <i>update odpoveď</i> “ | 15 |

Zoznam skratiek a značiek

API - rozhranie pre programovanie aplikácií
apt - Advanced Packaging Tool
CIA - dôvernosť, integrita, dostupnosť
CSV - čiarkou oddelené hodnoty
DAC - voliteľné riadenie prístupu
DTE - vynucovanie domény a typu
FEI - Fakulta elektrotechniky a informatiky
IAS - informačná dôvera a bezpečnosť
LSM - linuxové bezpečnostné moduly
MAC - povinné riadenie prístupu
NFS - sieťový súborový systém
OS - operačný systém
RBAC - riadenie prístupu na základe rolí
rwx - čítanie, zápis, spustenie
STU - Slovenská technická univerzita v Bratislave
TE - vynucovanie typu
UAC - riadenie prístupu používateľov
xattr - rozšírené atribúty
XML - rozšíriteľný značkovací jazyk

Úvod

Snáď ešte nikdy v histórii IT nebola počítačová bezpečnosť tak búrlivo diskutovanou témou, ako je tomu dnes. Môže za to viacero faktorov. Častokrát softvéroví vývojári odsúvali/odsúvajú bezpečnostný aspekt do úzadia s tým, že je nie príliš dôležitý vzhľadom na charakter ich softvéru - čo sa časom ukáže ako chybný predpoklad. K rozmachu tejto témy taktiež aktívne prispievajú užívatelia, ktorí sú väčšinou nedbanliví alebo len jednoducho nevedia ani o základných bezpečnostných princípoch - a niektorí dokonca urobia čokoľvek pre svoj komfort, vypínanie dôležitých bezpečnostných mechanizmov nevynímajúc. V neposlednom rade veľkou časťou túto diskusiu podnecujú útočníci - ľudia, ktorí využívajú vyššie uvedené okolnosti na súkromné obohatenie v akomkoľvek zmysle.

Softvérový vývoj je nikdy nekončiaci proces a preto so zvyšujúcou sa komplexitou programov narastá aj počet programátorských chýb v nich, OS nevynímajúc. Tento vývoj však môže znamenať aj posun k lepšiemu zabezpečeniu a oprave existujúcich chýb, avšak na druhej strane, čo bráni vývojárom hovoriť, že pridali novú vrstvu ochrany, keď všetko, čo spravili, bola zmena jedného makra v kóde a totálna ignorancia reálnej zraniteľnosti?

A práve tu vstupujú do hry doplnkové riešenia zabezpečenia. OS Linux je zraniteľný ako akýkoľvek iný OS, a preto, v niektorých prípadoch už viac ako dekádu, existujú doplnkové riešenia, ktoré zvyšujú mieru ochrany OS Linux (a tým aj používateľov). Jedným z týchto riešení je aj systém Medusa Voyager, nástupca systému Medusa DS9, v spojení s autorizačným serverom. Pre pôvodnú verziu Medusy existuje jediná implementácia autorizačného servera s názvom Constable.

Cieľom tejto práce je naprogramovať funkčný autorizačný server pre novú verziu Medusy, keďže ten pôvodný má viacero nevýhod a obsahuje niektoré závažné chyby. Táto práca je súčasťou snaženia členov vývojového tímu Medusa Voyager o poskytnutie funkčnej nadstavby zabezpečenia OS Linux pre užívateľov. Sekundárnym cieľom je snaha dostať Medusu do hlavnej vetvy vývoja Linuxového jadra, čo by značne mohlo akcelerovať jej vývoj a tým pádom aj rozšírenie tohto riešenia.

1 Analýza problematiky

Nasledujúca časť práce obsahuje základné vedomosti z oblasti bezpečnosti, vo všeobecnosti i v operačnom systéme Linux, ktoré sú nápomocné pri pochopení niektorých neskorších pasáží v texte.

1.1 Bezpečnosť

Pojem informačná bezpečnosť vo všeobecnosti enkapsuluje všetky postupy využívané na ochranu informácií pred neoprávneným prístupom k nim. To zahŕňa, okrem iného, ich používanie, odtažňovanie, zmenu, alebo zničenie.

Základnými kritériami používanými na vytyčovanie toho, čo považujeme za zabezpečené a čo už nie, sú tri nasledujúce pojmy: **dôvernosť**, **integrita** a **dostupnosť**. Tieto tri pojmy sú často v IT označované aj ako **CIA triáda** (akronym z angl. confidentiality, integrity, availability).

Dôvernosť je vo väčšine prípadov interpretovaná ako obmedzenie prístupu k informáciám len pre určitú skupinu ľudí. Vo väčšine prípadov sa preto informácie kategorizujú na rôzne úrovne prístupu podľa škôd, ktoré môže ich zneužitie/neoprávnený prístup k nim spôsobiť.

Integrita najväčšou oblasťou záujmu tejto práce. Ide o udržiavanie konzistentnosti údajov počas ich celého životného cyklu. To zahŕňa zamedzenie zmien neoprávnenými osobami (t.j. osobami, ktoré nemajú potrebnú dôvernosť) počas vytvárania, prenosu, úpravy alebo ničenia údajov. Táto časť modelu sa najčastejšie vynucuje správne nastavenými oprávneniami súborov (viď strana 3) a UAC.

Posledným pojmom z CIA triády je **dostupnosť**. Dostupnosť pojednáva skôr o hardvérovej stránke bezpečnosti a nie je predmetom záujmu tejto práce. Len pre úplnosť: dostupnosť sa zabezpečuje správnou údržbou hardvéru, OS a softvéru. Patrí sem napríklad predchádzanie softvérovým konfliktom, pravidelné aktualizovanie softvéru alebo OS a.i.

Postupom času začali k CIA triáde pribúdať ďalšie pojmy, pretože z dnešného pohľadu už nie je klasifikácia na „zabezpečené“ a „nezabezpečené“ taká jednoduchá, aká bola pred 20 rokmi, keď tento pojem vznikol. Nasledujú dve najznámejšie rozšírenia CIA triády.

V roku 2002 Donn Parker navrhol doplnkový model k CIA triáde, ktorý nazval *šesť atomických častí informácie* (six atomic elements of information), taktiež prezývanú aj *Parkeriánska hexáda*. K pojmom z CIA triády tak pribudli tri nové pojmy, menovite: vlastníctvo (possession), autenticita (authenticity) a užitočnosť (utility).

Ako posledný dodatok k pojmom v tejto oblasti dávame do pozornosti návrh, ktorý vychádza z podrobnej analýzy IAS literatúry. Je označovaný pojmom *IAS-oktáva* a bol navrhnutý ako logické rozšírenie CIA triády v roku 2013. Pozostáva z týchto pojmov: dôveryhodnosť (confidentiality), integrita (integrity), dostupnosť (availability), zodpovednosť (accountability), auditovateľnosť (auditability), dôveryhodnosť (authenticity/trustworthiness), nepopierateľnosť (non-repudiation) a súkromie (privacy).

Ako je vidieť, oblasť bezpečnosti v IT sa neustále vyvíja a prispôsobuje na základe nárastu požiadaviek na jemnejšiu segregáciu oprávnení a bezpečnejšie systémy pri zachovaní jednoduchej konfigurácii. A práve tu vstupujú do hry bezpečnostné politiky a k nim priliehajúca množina bezpečnostných modelov. Práve bezpečnostné modely sa snažia zjednodušiť implementáciu bezpečnostných politík v praxi.

1.2 Bezpečnostná politika a bezpečnostný model

Bezpečnostná politika riadenia prístupu špecifikuje zoznam činností a k nim priliehajúcich okolností, za ktorých je možné danú činnosť vykonať. Podstatou politiky je koncept trojice *operácia, subjekt, objekt*, nad ktorými politika vynáša jednoduché rozhodnutie: buď je táto trojica povolená, alebo zamietnutá.

Avšak určovať takýmito trojicami, ktoré operácie sú povolené a ktoré nie je veľmi rozsiahle a neprehľadné. Ďalším problémom takéhoto prístupu je napríklad aj nemožnosť rozhodnúť o subjekte, ktorý pred začatím operácie nad objektom neexistuje [9].

Kvôli týmto, a aj iným, dôvodom bol zavedený pojem bezpečnostného modelu, ktorý definuje ďalšie prvky: bezpečnostné úrovne, oddelenia, role a.i. Toto značne zjednodušuje definíciu a zmenu oprávnení podľa bezpečnostnej politiky konverziou trojice *operácia, subjekt, objekt* na zrozumiteľnejšie pojmy. Tieto pojmy následne môžu umožňovať napr. agregáciu objektov alebo operácií podľa spoločného prvku z bezpečnostného modelu (napr. role). No to zároveň znamená, že objekty aj subjekty v sebe musia niesť prídavné informácie, pomocou ktorých sa dajú priradiť k prvkom definovaným v bezpečnostnom modeli [9].

1.3 Bezpečnosť v OS Linux

Linux, operačný systém vychádzajúci z koncepcie Unixu, inšpirovaný Minixom, sa snaží zachovávať základný princíp Unixu: „Všetko je súbor. Čo nie je súbor, je proces.“ Presnejšie je však tvrdenie: „Všetko je popisovač súboru (file descriptor), alebo proces.“

Vďaka tomuto jednoduchému základu neboli v Unixe potrebné viaceré mechanizmy riadenia prístupu a programátori sa zamerali na riadenie prístupu k súborom. Nasledu-

júca sekcia popisuje toto riadenie v Unixe a systémoch na ňom založených, alebo ním inšpirovaných, kam patrí aj Linux.

1.3.1 Riadenie prístupu k súborom

Počas vývoja Unixu bol kladený dôraz na to, aby výsledný systém spĺňal tri základné kritéria: *prenositelnosť*, *schopnosť vykonávať viacero úloh v jednom čase* a *možnosť obsluhovať viacerých používateľov v jednom čase*. Tým vznikla potreba rozdeľovať používateľov podľa toho, k čomu môžu pristupovať a k čomu nie. Často je nevhodné až nežiadúce, aby mali všetci používatelia prístup ku všetkým zdrojom. Preto vznikol koncept, ktorý delí oprávnenia v Unixových systémoch do troch kategórií: separátne práva pre **vlastníka** (owner), pre ľudí zo **skupiny** (group), v ktorej sa vlastník nachádza a kategória **ostatní** (others), niekedy aj označovaná súhrnným názvom *svet*.

Každá z týchto kategórií má pridelené tri operácie, pomocou ktorých OS rozhoduje, či používateľ z danej kategórie má právo nad objektom vykonať danú operáciu, alebo nie. Tieto operácie sú: **čítanie** (read), **zápis** (write) a **vykonanie** (execute). Práva sa zvyčajne zapisujú v osmičkovej sústave, napr. 755, kde každá cifra prislúcha jednej z troch kategórií a jedna cifra obsahuje nastavenia pre tri definované operácie.

Nutné podotknúť, že vo všetkých Unixových systémoch a systémoch im podobným existuje rola tzv. superpoužívateľa, označovaného tiež ako root. Na tohto používateľa sa vyššie uvedený bezpečnostný model nevzťahuje. Tento používateľ tak môže vykonávať akékoľvek privilegované či neprivilegované operácie v systéme, čo je potenciálnym bezpečnostným rizikom.

V Linuxe navyše k oprávneniam z Unixu pribudli tzv. **POSIX schopnosti** (POSIX capabilities). Tie majú za úlohu rozdeliť práva správcu na viacero neprekrývajúcich sa skupín. Cieľom tohto opatrenia je zamedzenie existencie jednej entity s absolútnou kontrolou nad systémom. V súčasnej verzii linuxového jadra sú tieto oprávnenia uchovávané pomocou *xattr*, rozšírených atribútov súborov. To znamená, že sú použiteľné len na súborových systémoch, ktoré tieto atribúty vedia spracovať a uchovávať (EXT4, Btrfs, ReiserFS, JFS a ZFS). Avšak vzhľadom na nerovnomerné rozdelenie „sily“ jednotlivých schopností nie sú až takým účinným mechanizmom na ochranu, akým by sa na prvý pohľad mohli javiť. Problémovou môže byť v linuxových systémoch napr. schopnosť zapisovať do akéhokoľvek súboru, čím efektívne získavate kontrolu nad celým systémom.

1.4 Doplnkové riešenia zabezpečenia

Po predstavení základných bezpečnostných konceptov zo sveta OS Linux sa zameriame na riešenia, ktoré umožňujú používateľom zvýšiť mieru zabezpečenia ich operačného

systému. Od vydania pôvodnej verzie Medusy DS9 uplynulo už viac ako 15 rokov a vývoj na tomto poli priniesol veľa vylepšení do dlhotrvajúcich projektov a pár úplne nových riešení.

1.5 Riešenia založené na LSM

V dobe, keď vyšla prvá Medusa, ešte neexistovala knižnica LSM, na ktorej je postavená väčšina zo súčasných bezpečnostných riešení. LSM knižnica predstavuje spoločné API pre programátorov, ktorí chcú vyvíjať bezpečnostné moduly pre jadro [5]. Cieľom tejto knižnice je zabezpečiť prístup k prostriedkom, ktoré sú potrebné pre implementáciu MAC modelu, a to bez výrazných zásahov do jadra Linuxu. LSM funguje na princípe vkladanie API hákov (API hook) do každého systémového volania v bode, kedy sa má pristúpiť k dôležitému objektu z jadra, napr. k i-uzlu (inode). Tento hák následne preruší vykonávanie daného volania a predá riadenie niektorej z funkcií bezpečnostného modulu. Tá následne rozhodne o prístupe k objektu. Po vykonaní prerušenia sa vráti kontrola jadru a to pokračuje ďalej v behu. V prípade, že v jadre nie je zavedený žiadny doplnkový bezpečnostný modul, rozhodovanie vykonáva fingovaný bezpečnostný modul dodávaný s OS. Modul funguje na klasickej superpoužívateľskej logike. LSM knižnica je štandardnou súčasťou linuxového jadra od verzie 2.6 [6] a jej použitie je základným predpokladom k zaradeniu modulu do hlavnej vetvy vývoja linuxového jadra.

1.5.1 SELinux

Nepopierateľne najznámejší modul, pochádzajúci od NSA a firmy Red Hat, je implementáciou MAC pre Linux. Oficiálne sa nachádza v jadre od verzie 2.6.0-test3. Jeho základ tvorí FLASK, vďaka čomu tento modul podporuje veľa MAC politík, vrátane vynútenia typu (type enforcement), RBAC modelu a viacúrovňovej bezpečnosti. Bežiaci SELinux pracuje na princípe *bieleho listu* (white listing), keď pri každej požiadavke odopiera prístup ku všetkým zdrojom okrem tých, ktoré sú explicitne povolené [4].

Používatelia na úrovni SELinuxu nemusia korelovať s klasickými používateľmi. Mapovanie SELinux používateľa na linuxový účet s rovnakým menom je však často využívané riešenie, kvôli prehľadnosti. Jeden SELinux používateľ môže byť namapovaný na viacero linuxových účtov, ale naopak to neplatí. Každý linuxový účet musí byť namapovaný na práve jedného SELinux používateľa. Každý SELinux používateľ následne môže mať priradených viacero rolí.

Najpodstatnejšou politikou, prítomnou pri akejkoľvek zmysluplnej konfigurácii, je

vynútenie typu, ktoré funguje na systéme *nálepiek* (labels) využívajúcich, narozdiel od iných modulov, i-uzly. Každá nálepka obsahuje množinu informácií (používateľ, rola, meno, a ďalšie), medzi ktorými sa nachádza aj *typ*. Každý zdroj (či už objekt, alebo subjekt) v systéme obsahuje takúto nálepku. Zdroje, ktoré zdieľajú typ alebo viaceré časti nálepky sú považované za rovnocenné a zaobchádza sa s nimi rovnako.

1.5.2 AppArmor

AppArmor je bezpečnostný modul častokrát porovnávaný s SELinuxom, najskôr kvôli jeho rozšírenosti (nachádza sa v základnej inštalácii Ubuntu a openSUSE). Pôvodne bol tento MAC modul vytvorený ako súčasť bezpečnej linuxovej distribúcie Immunix a volal sa SubDomain. Oficiálne sa v jadre nachádza od verzie 2.6.36. Momentálne je vyvíjaný firmou Canonical Ltd., známou ako tvorcom Ubuntu Linuxu. Hlavnou myšlienkou tvorcov modulu bola jednoduchá konfigurácia a nasadenie. Ponúka sa priame porovnanie so SELinuxom, ktorý je oveľa viac komplexný - a teda aj zložitejší na správne nakonfigurovanie a nasadenie. Pravidlá v AppArmore sú, podobne ako v SELinuxe, typu „biely list“. Na úrovni systému však môže AppArmor fungovať v režime bieleho listu (v prípade nasadenia na celý systém) alebo čierneho listu (v prípade inkrementálneho nasadenia). Časť systému sledovaná modulom ale vždy podlieha politike bieleho listu.

AppArmor má dva základné režimy prevádzky. Učiaci režim, v ktorom zapisuje všetky neoprávnené prístupy ku zdrojom, slúži na vytvorenie konfigurácie neskúseným užívateľom (na základe vytvorených logov ponúka pravidlá pre biely list). V režime vynucovania politiky už následne na základe definovaných pravidiel rozhoduje pomocou DTE mechanizmov o zdrojoch v systéme [3].

V porovnaní so SELinuxom využíva AppArmor iný prístup k označovaniu zdrojov. Každý zdroj je jednoznačne identifikovaný svojou cestou v súborovom systéme. To má za následok nezávislosť na súborovom systéme (SELinux potrebuje súborový systém podporujúci bezpečnostné nálepky). Pomocou tohto modulu teda môže administrátor spravovať aj sieťové médiá pripojené pomocou NFS. Tento prístup však má aj svoje negatívne stránky, napr. pri vytvorení novej tvrdej linky na súbor sa obchádzajú oprávnenia nastavené na pôvodnom súbore, čo môže byť problém pri inkrementálnom nasadení.

1.5.3 Smack

Jedna z novších implementácií bezpečnostného modulu pre Linux. Hlavným cieľom pri návrhu tohto MAC modulu bola jednoduchosť v porovnaní s inými modulmi. Smack je oficiálnou súčasťou jadra od verzie 2.6.25 a je od počiatku vyvíjaný nezávislým autorom, ktorým je Casey Schaufler.

Smack pozostáva z troch častí. Prvou z nich je modul v kerneli OS. Pre najlepšiu funkčnosť autor odporúča použitie súborového systému s podporou xattr, ale to nie je nutnosť. Druhou časťou je sada nástrojov/záplat, ktoré slúžia na správnu interpretáciu Smack atribútov súborovým systémom. Tretou časťou je spúšťač skript, ktorý načítava konfiguráciu modulu a zároveň zabezpečuje, že súbory majú správne nastavené Smack atribúty. Konfiguračné súbory sa štandardne nachádzajú vo vlastnom pseudo súborovom systéme (smackfs) [7].

Striktne vzaté, Smack neimplementuje žiadnu konkrétnu politiku. Základným konceptom bolo zobrať si to najlepšie z už existujúcich politík a spojiť ich do koherentného celku. V Smacku sa tak nachádzajú vlastnosti modelu Bell-LaPadula a iných, pričom žiadne z nich buď nie sú implementované v plnom rozsahu, alebo sú vyriešené ich hlavné nevýhody.

Modul využíva systém nálepiek podobný tomu implementovanému v SELinuxe, kde hlavným rozdielom je používanie absolútnych ciest (podobne ako v AppArmor) a absencia (D)TE. Absencia (D)TE je dosiahnutá definíciou kontroly prístupu v zmysle operácií, ktoré sú už v Linuxe zadefinované (rwx).

1.5.4 TOMOYO

Posledným zo série štandardných bezpečnostných modulov, nachádzajúcich sa v hlavnej vývojovej vetve jadra (od verzie 2.6.30), je MAC modul TOMOYO. Celkom zábavný je fakt, že modul je nazvaný podľa kreslenej postavičky. Pôvodným autorom tohto modulu je firma NTT Data Corporation, ktorá projekt podporovala do roku 2012.

Základným bezpečnostným modelom tohto modulu je DTE. Na identifikáciu zdrojov sú použité absolútne cesty (podobne ako AppArmor) a každý zdroj má vlastnú bezpečnostnú nálepku. TOMOYO v sebe neobsahuje koncepty používateľov a rolí tak, ako to robí napr. SELinux. V systéme existuje len administrátor, ktorý má možnosť meniť konfiguráciu a delegovať administrátorské úlohy, a ostatní používatelia, ktorí podliehajú vytvorenej politike. Základom konfigurácie sú domény, do ktorých patria zdroje. Zdroje môžu na základe zmeny svojho stavu meniť doménu, do ktorej patria (toto platí pre procesy). Vývojári tohoto modulu to nazývajú zmenou stavu (state transition), v podstate sa však jedná o TE. Každá doména má následne svoj vlastný profil. Ten obsahuje povolené akcie, ktoré môžu zdroje v danej doméne vykonávať. Aktualizácia pravidiel pre profil prebieha v reálnom čase, nie je treba reštartovať procesy kvôli aplikovaniu vykonaných zmien v konfigurácii. Reštart procesov je potrebný iba v prípade, že správca zmaže doménu a vytvorí ju nanovo [8].

TOMOYO v sebe zahŕňa samoučiaci mechanizmus, podobný tomu z AppArmoru.

Modul sleduje prístupy ku zdrojom a na ich základe vytvára konfiguračné pravidlá nutné pre normálny beh aplikácií. Toto je jednou zo silných stránok tohto modulu, ktorá umožňuje rýchly prechod na tento systém aj začínajúcemu správcovi (na rozdiel od SELinuxu).

1.6 Riešenia bez použitia LSM

Aj cez to, že použitie LSM knižnice je hlavným predpokladom na začlenenie riešenia do hlavnej vetvy vývoja jadra, existujú riešenia, ktoré služieb tejto knižnice nevyužívajú. Nižšie je uvedený najznámejší projekt tohto druhu.

1.6.1 grsecurity

Grsecurity je doplnkové bezpečnostné riešenie pre linuxové jadro. Skladá sa z viacerých častí, z ktorých je väčšina plne funkčná aj sama osebe. Korene projektu siahajú do roku 2001, kedy bol vytvorený ako klon záplat pre linuxové jadro z projektu Openwall. Prvé vydanie grsecurity bolo určené pre jadro verzie 2.4.1.

Základnou súčasťou riešenia je záplata PaX, ktorá, okrem iného, označí pamäť dát (napr. zásobník) ako nespustiteľnú a pamäť programu ako nezapisovateľnú. Okrem toho táto záplata vykoná znáhodnenie rozloženia adresného priestoru (ASLR). Znamená to, že rozistribuuje časti programu na vzájomne nesúvisiace adresy v pamäti. Tieto dve opatrenia veľmi zúžia možnosti útoku na daný systém, napr. vylúčia isté druhy pretečenia zásobníka. PaX je samostatne vyvíjaný projekt.

Ďalšou podstatnou súčasťou grsecurity je RBAC systém. Základnou myšlienkou RBAC modelu je princíp najmenších privilégií (každý objekt musí byť schopný pristupovať len ku zdrojom, ktoré sú nevyhnutné pre jeho úlohu) a separácia povinností (na dokončenie úlohy sa vyžaduje viac ako jedna osoba). Na rozdiel od MAC/DAC systémov, kde sú povolenia priradené k používateľským identitám, pri RBAC sú povolenia priradené priamo rolám. Oddelená politika následne priraduje možné role k používateľom. Používatelia si následne vyberajú role, ktoré chcú používať, a to pri každom prihlásení [6].

Grsecurity poskytuje aj ďalšie záplaty, napr. obmedzenie dosahu chroot operácie (žiadny prístup k zdieľanej pamäti mimo chroot, žiadny kill, ptrace mimo chroot a.i.), zákaz vytvárania tvrdých odkazov na súbory, ktoré užívateľ nevlastní a ďalšie.

2 Medusa Voyager

Medusa Voyager je pokračovateľom projektu Medusa DS9, ktorý v rokoch 1997-2001 založili na FEI STU Marek Zelem a Milan Pikula. Primárnym cieľom projektu Medusa Voyager bolo preniesť pôvodnú Medusu DS9 spolu s autorizačným serverom Constable (32-bitové verzie) na nové linuxové jadro (64-bitové verzie), čo sa v roku 2014 podarilo Jánovi Káčerovi [1]. Ďalším z významných cieľov projektu bol prechod na LSM, ktorý by mohol Meduse umožniť dostať sa do hlavnej vetvy vývoja jadra a potenciálne tak získať komunitu programátorov, ktorí by sa na projekte aktívne podieľali. V súčasnosti prebieha opätovný vývoj na FEI s cieľom pridať ďalšie systémové volania do modulu bežiaceho v jadre.

Jedným z dôležitých bodov pre zahrnutie Medusy do hlavnej vetvy vývoja jadra je, okrem využívania LSM modulu, aj implementácia plne funkčnej verzie autorizačného serveru.

Medusa Voyager je riešenie, skladajúce sa z dvoch hlavných častí: monitorovací modul, bežiaci v jadre, a autorizačný server, bežiaci v užívateľskom priestore. Tieto dve časti sú od seba vzájomne nezávislé, čo vnímame ako najväčšiu výhodu tohto bezpečnostného riešenia.

Vďaka tejto unikátnej konštrukcii v jadre beží len istá časť kódu. Tá má za úlohu sledovať dianie v systéme a na základe tohto diania posilať žiadosti autorizačnému serveru. Samotný autorizačný server beží v užívateľskom priestore a rozhoduje o žiadostiach, ktoré mu modul z jadra posila. Na základe odpovede následne modul v jadre povolí alebo odoprie prístup k danému zdroju.

Pretože modul bežiaci v jadre je nezávislý na autorizačnom serveri, je možné autorizačný server kedykoľvek vymeniť za iný. Ak teda koncový užívateľ bude chcieť nasadiť napr. RBAC model namiesto MAC, môže tak kedykoľvek spraviť, bez zásahov do zdrojových kódov modulu v jadre.

2.1 Komunikačný protokol

Medusa komunikuje s autorizačným serverom pomocou znakového zariadenia */dev/medusa*. Pomocou tohto zariadenia si modul s autorizačným serverom navzájom vymieňajú binárne správy. Tieto správy možno rozdeliť na tri skupiny. Prvá skupina sú správy inicializačné. Sem patrí uvítanie, ktoré je istou formou handshake, a správy o systéme podporovaných entitách. Medusa posila tieto správy ihneď po spustení, aj bez prítomnosti aktívneho autorizačného servera. Druhá skupina správ je autorizačná. Sem patria

správy so žiadosťami o autorizáciu a odpovede na ne. Medusa vždy posiela len jednu autorizačnú správu v danom časovom okamihu. Ďalšie žiadosti o autorizáciu čakajú v rade na spracovanie predchádzajúcej žiadosti. Tretou skupinou sú režijné správy: fetch a update. Tieto správy posiela autorizačný server Meduse v prípade, že potrebuje detailnejšie informácie o entitách vystupujúcich v autorizačnej žiadosti. Schematický náhľad komunikácie sa nachádza na Obrázku 1.

Podporované entity sa delia na dve triedy: k-objekty a udalosti.

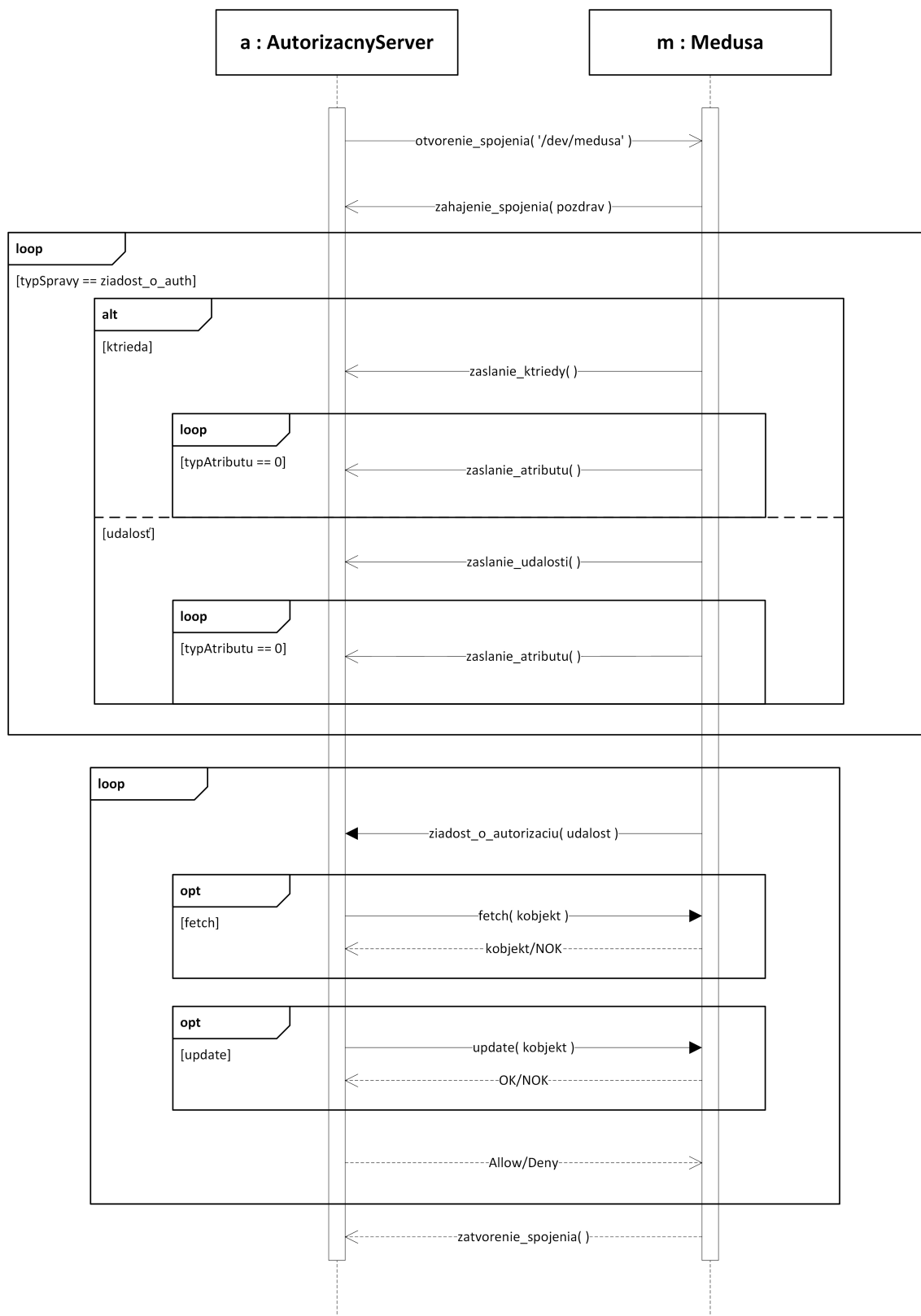
K-objekty sú všetky významné zdroje, o ktorých chceme rozhodovať. Môžu vystupovať v rolách objektov aj subjektov, čo záleží na type udalosti. Napr. žiadosť s udalostou typu syscall (systémové volanie, slúži na posielanie signálov medzi procesmi) obsahuje 2 k-objekty typu process. Jeden vystupuje v roli objektu (proces, ktorý signál posiela), druhý v roli subjektu (proces, ktorý signál prijíma). Nad každým k-objektom by mali byť kvôli správnej funkčnosti zadané funkcie fetch a update, ktoré sa používajú na aktualizáciu stavu k-objektov. V niektorých prípadoch nemusia byť implementované obe (pre printk napr. fetch nemá zmysel), ale obe by mali existovať.

Udalosti sú operácie, ktoré môžu byť nad definovanými k-objektami vykonávané a autorizačný server by na ne mal vedieť náležite reagovať. Existujú dve udalosti, ktoré slúžia pre interné potreby komunikácie medzi Medusou a Constablom: getprocess a getfile. Pred tým, než z jadra príde žiadosť o autorizáciu, Medusa zistí, či už autorizačný server pozná k-objekty, ktoré vystupujú v žiadosti. Toto Medusa vie pomocou verziovania k-objektov, kde každý k-objekt má pri prvotnej inicializácii nastavený indikátor verzie na 0. Medusa potom pomocou porovnávania indikátoru verzie s poradovým číslom pripojenia vie, či autorizačný server daný k-objekt pozná a má jeho aktuálnu verziu. Ak sú čísla zhodné, autorizačný server má správne k-objekty už načítané a Medusa mu zašle len žiadosť o autorizáciu. V prípade, že sa tieto čísla líšia, Medusa zašle autorizačnému serveru k-objekty vystupujúce v udalosti pomocou udalosti getprocess (ak sa jedná proces) alebo getfile (ak sa jedná súbor alebo rozhranie) a žiadosť o autorizáciu udalosti prichádza potom. Tento mechanizmus slúži ako ochrana pred zasielaním udalostí s neznámymi k-objektami.

V nasledujúcej časti sú stručne popísané jednotlivé typy správ.

2.1.1 Pozdrav

Na začiatku znakového zariadenia sa nachádza pozdrav, ktoré slúži na určenie endiannessy. V súčasnosti táto správa nie je veľmi dôležitá, pretože existujú aj jednoduchšie spôsoby zistenia endiannessy. Ak by však bol v budúcnosti implementovaný pôvodný koncept s možnosťou sieťovej autorizácie, kde na cieľovom stroji môže byť iný endian ako



Obrázok 1: Komunikačný diagram

stroji s autorizačným serverom, bude aj táto správa potrebná. Formát správy je uvedený v Tabuľke 1.

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|----------|
| uint | 8 | GREETING |

Tabuľka 1: Formát správy „pozdrav“

2.1.2 Inicializačné správy

Podporované entity sú zasielané po samotnom pozdrave. Štruktúra správy sa líši na základe toho, či ide o k-objekt (Tabuľka 2) alebo o udalosť (Tabuľka 3). Tieto správy slúžia ako návod (loader) na čítanie k-objektov a udalostí pre autorizačný server, prichádzajúcich v autorizačnej fáze. Toto je dôležité, pretože nie na všetkých systémoch existujú tie isté štruktúry v jadre. Taktiež sa medzi rôznymi verziami jadra môžu dve zhodne volajúce štruktúry (k-objekty) líšiť, napr. atribútmi.

Pri položke *atribúty* v Tabuľkách 2 a 3 reprezentuje písmeno *N* počet atribútov.

| Dátový typ | Veľkosť [byte] | Obsah |
|-------------|----------------|-----------|
| uint | 8 | - |
| uint | 4 | CLASS_DEF |
| uint | 8 | Medusa ID |
| uint | 4 | dĺžka |
| char[] | 30 | meno |
| attribute[] | N*32 | atribúty |

Tabuľka 2: Formát správy „podporovaný k-objekt“

2.1.3 Autorizačné správy

Tieto správy nasledujú po inicializačnej fáze. Žiadosti o autorizáciu sú reakciami na pokus o pridelenie istého zdroja objektu.

Druhým typom správ sú autorizačné odpovede, ktoré sú reakciou na autorizačné žiadosti. Obsahujú rozhodnutie, vynesené bezpečnostnou politikou autorizačného servera.

Typický príklad: pokúsite sa vytvoriť nový súbor. Medusa odošle autorizačnému serveru žiadosť o vytvorenie i-uzla na disku. Autorizačný server správu prijíme, vyhodnotí podľa politiky, či už kladne alebo záporne. Toto rozhodnutie následne autorizačný server pošle Meduse, ktorá ho v jadre vynúti.

Formáty jednotlivých správ sú podrobne opísané v Tabuľke 5 (žiadosť o autorizáciu)

| Dátový typ | Veľkosť [byte] | Obsah |
|-------------|----------------|---------------------------------------|
| uint | 8 | - |
| uint | 4 | EVENT_DEF |
| uint | 8 | Medusa ID |
| uint | 4 | dĺžka |
| uint | 4 | príznak akcie |
| uint[] | 2*8 | Medusa ID pre operandy (k-objekty) |
| string | 30 | meno |
| char[] | 2*27 | mená operandov |
| attribute[] | N*32 | atribúty udalosti |

Tabuľka 3: Formát správy „podporovaná udalosť“

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|--------|
| uint | 4 | offset |
| uint | 4 | dĺžka |
| uint | 2 | typ |
| char[] | 27 | meno |

Tabuľka 4: Štruktúra dátového typu „attribute“

a 6 (odpoveď). Veľkosť udalosti (dátový typ event) a k-objektov (dátový typ kobject) sa líšia v závislosti na type udalosti, resp. k-objektu. Ich presná veľkosť sa nachádza v správach o podporovaných entitách v premennej *dĺžka* (Tabuľky 2 a 3).

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|----------------------|
| uint | 8 | Medusa ID udalosti |
| uint | 4 | ID žiadosti |
| event | ?? | udalosť |
| kobject[] | ?? | k-objekty (operands) |

Tabuľka 5: Formát správy „*autorizačná žiadosť*“

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|---------------|
| uint | 8 | AUTH_ANS |
| uint | 4 | ID žiadosti |
| uint | 2 | návratový kód |

Tabuľka 6: Formát správy „*autorizačná odpoveď*“

2.1.4 Režijné správy

V tejto sekcii sa nachádzajú správy *fetch* (Tabuľky 7 a 8) a *update* (Tabuľky 9 a 10). Štandardne sa tieto správy posielajú medzi žiadosťou o autorizáciu a odpoveďou na ňu, ak sa zmenil stav k-objektu. V prípade, že nemáme čo aktualizovať na stave k-objektov (nič sa nezmenilo) nie je nutné tieto správy posilať. Rozdiel medzi správami je nasledovný: v prípade správy typu *fetch* si autorizačný server pýta od jadra dodatočné údaje pred odpoveďou na žiadosť, kým v prípade správy typu *update* žiada, aby bol stav objektu aktualizovaný v jadre, pretože ho autorizačný server zmenil.

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|---------------------|
| uint | 8 | FETCH_REQ |
| uint | 8 | Medusa ID k-objektu |
| uint | 8 | ID žiadosti |
| kobject | ?? | k-objekt |

Tabuľka 7: Formát správy „*fetch žiadosť*“

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|---------------------|
| uint | 8 | - |
| uint | 4 | FETCH_ANS |
| uint | 8 | Medusa ID k-objektu |
| uint | 8 | ID žiadosti |
| kobject | ?? | k-objekt |

Tabuľka 8: Formát správy „*fetch odpoved*“

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|---------------------|
| uint | 8 | UPDATE_REQ |
| uint | 8 | Medusa ID k-objektu |
| uint | 8 | ID žiadosti |
| kobject | ?? | k-objekt |

Tabuľka 9: Formát správy „*update žiadosť*“

| Dátový typ | Veľkosť [byte] | Obsah |
|------------|----------------|---------------------|
| uint | 8 | - |
| uint | 4 | UPDATE_ANS |
| uint | 8 | Medusa ID k-objektu |
| uint | 8 | ID žiadosti |
| kobject | ?? | k-objekt |

Tabuľka 10: Formát správy „*update odpoved*“

3 ReStable

Táto sekcia opisuje samotné riešenie, ktoré je výsledkom činnosti na tejto práci. Časť riešenia s komunikačným protokolom priamo vychádza z autorizačného servera Constable od Mareka Zelema [9]. Ten sa v priebehu času veľmi nezmenil (drobné úpravy však nastali a riešenie tejto práce ich reflektuje). Zvyšok programového kódu je vlastná implementácia od autora tejto práce.

3.1 Špecifikácia požiadaviek

Nový autorizačný server pre Medusu Voyager by mal spĺňať nasledujúce požiadavky:

1. platformová nezávislosť
2. modulárny prístup
3. výpočtovo rýchla implementácia
4. povoľovanie všetkých aktivít
5. zaznamenávanie prístupov k objektom v systéme

Platformová nezávislosť je pre tento projekt veľmi dôležitá, a to kvôli možnosti spustiť Medusu Voyager s autorizačným serverom pod akoukoľvek distribúciou Linuxu.

Pomocou modulárneho prístupu jednoducho docielime vzájomnú spoluprácu modulov aj v prípade, že ich písali rôzni autori, za predpokladu, že používajú definovaný interface. Tento bod je dôležitý vzhľadom na rozšírenia riešenia v neskorších fázach projektu, napr. o rozhodovacie moduly, kde každý modul môže implementovať iný bezpečnostný model.

Záujem o výpočtovo rýchlu implementáciu je pri riešení tohto typu samozrejímavý. Na to, aby bola Medusa s autorizačným serverom reálne využiteľná, musíme minimalizovať dopad na užívateľský komfort. Akékoľvek spomalenie systému je užívateľom vnímané veľmi negatívne a podpisuje sa na ochote využívať dané riešenie.

V tejto základnej verzii nebude autorizačný server obsahovať žiadnu rozhodovaciu logiku. Namiesto toho bude povoľovať všetky akcie. Kód však bude štruktúrovaný tak, aby bolo kedykoľvek rozšíriť autorizačný server o MAC, RBAC alebo iný bezpečnostný model.

Poslednou požiadavkou je, aby autorizačný server zaznamenával všetky žiadosti o prístup k objektom v systéme. Toto by v budúcnosti mohol byť základ pre samoučiaci mechanizmus, vytvárajúci si pravidlá podľa záznamov.

3.2 Návrh

Na programovanie nového autorizačného servera je vhodné využiť niektorý s vyššie úrovňových modulárnych jazykov, ktoré sú multiplatformové. Zlepší sa tým čitateľnosť kódu, kvôli menšiemu počtu riadkov a jednoduchším konštruktom. Taktiež bude týmto krokom zaistený bezproblémový prenos kódu medzi strojmi s rôznymi architektúrami bez zmien v implementácii, alebo nutnosti rekompilácie.

Pôvodný návrh autorizačného servera spočíval v rozdelení servera do 4 významných modulov:

1. modul **kobject**
2. modul **event**
3. modul **msg_handler**
4. modul **logger**

Počas implementácie sme však zistili, že tento model nie je až tak vhodný. Preto došlo k závažnej zmene štruktúry (a mena) dvoch modulov: modulu **kobject** a modulu **event**. Prišli sme na to, že bude vhodnejšie zjednotiť načítanie pomocou abstraktnej nadtriedy. Taktiež sa ukázalo ako vhodné zjednotiť vytváranie dynamických tried, kde pre správnu funkčnosť plne postačuje jedna dynamická trieda pre vytváranie k-objektov aj udalostí. Ďalej pribudol nový modul, držiaci konštanty využívané v autorizačnom serveri.

Pôvodná štruktúra sa tak zmenila na nasledujúcu:

1. modul **loader**
2. modul **entity**
3. modul **msg_handler**
4. modul **logger**
5. modul **constants**

V tejto verzii je autorizačný server ľahko rozšíriteľný o ďalšie moduly len pomocou interakcie s modulom *msg_handler*, v ktorom sa nachádza celá aplikačná logika.

3.2.1 Modul loader

Modul loader obsahuje triedy, ktoré majú za úlohu udržiavať postup pre správne načítanie k-objektov a udalostí. Majú spoločnú abstraktnú nadtriedu a obe triedy obsahujú slovník pre rýchle vyhľadávanie v načítaných návodoch na čítanie prichádzajúcich entít. Každá inštancia následne obsahuje kompletné informácie potrebné pre načítanie konkrétnej entity do programu, alebo jej zápis v správnom formáte do znakového zariadenia.

3.2.2 Modul event

Modul entity obsahuje jednoduchú triedu s dynamickým konštruktorom. Ten pomocou predaného slovníka vybuduje inštanciu s atribútmi, ktoré sa nachádzajú v slovníku. Táto trieda nemá v súčasnosti žiadne ďalšie metódy a slúži len ako kontajner pre jednoduchšiu prácu s načítanými entitami. Tento prístup je nutný, pretože v čase kompilácie nevieme, aký typ k-objektu, alebo udalosti, s akými atribútmi nám počas behu autorizáčného servera príde.

3.2.3 Modul msg_handler

Modul msg_handler obsahuje jadro celého autorizáčného servera. Umožňuje nízkoúrovňový prístup k znakovému zariadeniu `/dev/medusa`, z ktorého je schopný čítať a zapisovať do neho. Modul funguje sekvenčne, t.j. načítava správy po jednej. Na základe získanej správy následne zvolí jednu z dostupných akcií na vykonanie nad touto správou. Vo všeobecnosti proces pozostáva z parsovania informácií a ich následného spracovania v hlavnom cykle programu. Spracovanie sa líši na základe typu správy.

Samotný modul je rozdelený na dve časti. Prvá časť je reprezentovaná triedou *Postman*, ktorá slúži ako interface medzi zvyškom modulu a znakovým zariadením. Druhá časť modulu je trieda *MessageHandler*. V tejto triede sa nachádza logika pre vyhodnotenie endianu, načítanie podporovaných entít, ich spracovanie, načítanie žiadostí o autorizáciu a odpovedanie na ne.

3.2.4 Modul logger

Modul logger slúži na zaznamenávanie prístupov k objektom v systéme. Slúži vytváranie logov za behu programu a na ich následné preliatie do samostatného súboru pri ukončení autorizáčného servera. Modul je konštruovaný štýlom abstraktnej triedy, v ktorej sa v súčasnej dobe nachádza implementácia exportu záznamov do CSV súboru. V budúcnosti tak bude možné vďaka tomuto riešeniu tento modul jednoducho rozšíriť. Rozšírením môže byť napríklad podpora iných formátov (XML), a to jednoduchou implementáciou abstraktnej nadtriedy.

3.2.5 Modul constants

V implementácii taktiež pribudol jeden nový modul: `constants`. Tento modul, ako už názov napovedá, obsahuje konštanty nutné pre správnu funkčnosť celého autorizačného servera, najmä konštanty používané v správach. Taktiež obsahuje konštantu `DEBUG`, ktorá vypína alebo zapína debuggovacie výpisy. V základe je táto možnosť vypnutá.

3.3 Implementácia riešenia

Pre naše účely sme si zvolili Python vo verzii 3, ktorý je interpretovaným vysokoúrovňovým programovacím jazykom. Týmto zabezpečíme splnenie prvého bodu, a síce platformovú nezávislosť. Samotný Python je koncipovaný štýlom modulárneho jazyka, čo nám dalo výhodu pri písaní kódu a zjednodušilo splnenie bodu číslo dva.

Kvôli zrýchleniu vykonávania programu boli preferované niektoré štruktúry programovacieho jazyka Python pred inými. Pri implementácii riešenia nastalo viacero situácií, v ktorých sa ponúkali rôzne cesty na vyriešenie situácie. Ako príklad nech nám poslúži vyhľadávanie v inštanciách triedy *Loader* z modulu *loader*. Prvým riešením bolo použiť list a *list comprehension* na vyhľadávanie. Druhým riešením bolo použiť list a využiť metódu `filter` s lambda výrazom. Tretím riešením bolo iterovať celým listom, až kým nenájdeme hľadané riešenie. Štvrtým riešením bolo použitie slovníka s vhodnými kľúčmi. Existujú aj iné riešenia tohto problému, my sa však obmedzíme na tieto. Každý z týchto prístupov funguje. V čom sa však líšia je ich časová zložitosť.

Časová zložitosť vyhľadávania v listoch je priamo úmerná ich veľkosti. Preto vyhľadávanie v liste je v akomkoľvek prípade pomalšie, ako vyhľadávanie v slovníku. Slovník využíva svoje kľúče na výpočet hashu. Následné vyhľadávanie v kľúčoch tak má časovú zložitosť $O(1)$. Daňou za to ale je väčšie využitie iných zdrojov, menovite operačnej pamäte. Keďže moderné počítače obsahujú rádovo gigabajty operačnej pamäte (a pamäť teda nie je problémom), zvolili sme v tomto prípade riešenie využívajúce slovník [2].

Okrem využitia rýchlych štruktúr odporúčame na zvýšenie výkonu použiť iný implementácia Pythonu ako klasický CPython, napr. PyPy.

3.4 Overenie funkčnosti a problémy

Overenie funkčnosti riešenia prebiehalo na bežnom desktopovom počítači (Core i5 3470, 16GB RAM, 256GB SSD disk, nVidia GeForce 750Ti) v prostredí VirtualBox s nainštalovaným OS Debian testing, jadro verzie 4.4.0. ReStable bol nastavený na automatické spustenie po prihlásení používateľa v režime bez debuggovacích výpisov so zapnutým logovaním. Súčasťou prílohy na CD nosiči je aj CSV log z polhodinového

používania počítača.

Samotné testovanie spočívalo v bežnom používaní počítača po dobu jedného dňa. Počas tejto doby boli na počítači vykonávané bežné činnosti: surfovanie po internete, púšťanie videoklipov z webových stránok, editovanie textových dokumentov, hranie nenáročných počítačových hier. Pre otestovanie prípadného úzkeho hrdla systému bol spustený update programov pomocou príkazu *apt-get upgrade*, zároveň s ním bolo spustené prehrávanie streamovaného videa, zapnutá počítačová hra a na pozadí bežalo ripovanie pesničiek z CD do bezstratového formátu.

Počas bežného používania prakticky nebolo možné postrehnúť výkonový pokles. Autorizačný server fungoval bez pádov a aplikácie bežali plynulo a bez obmedzení.

Počas záťažového testu bol systém testovaný dvakrát. Prvýkrát bol test zahájený na systéme s normálnym jadrom bez Medusy. Následne bol urobený rollback systému do pôvodného stavu a test bol zopakovaný s Medusou v jadre a bežiacim serverom ReStable. Výkonový rozdiel bol merateľný a podľa meraní sa pohyboval niekde na hranici 5-10% poklesu výkonu. Tento odhad vznikol na základe porovnávania trvania procesu ripovania CD nosiča a aktualizácie programov pomocou *apt*. Zavedenie bezpečnostnej politiky s najväčšou pravdepodobnosťou tento rozdiel ešte viac prehĺbi.

V priebehu implementácie sme narazili na špecifický problém, spojený s udalosťou **getprocess**. Táto udalosť má v loaderi uvedené, že so sebou nesie 2 k-objekty, čo je však chyba. ZP framework totiž definuje, že udalosti, v ktorých majú oba k-objekty zhodné mená, obsahuje len jeden takýto k-objekt. Chyba bola spôsobená pokusom čítať druhý k-objekt za koncom správy. To v súčinnosti s blokovacím režimom znakového zariadenia spôsobilo čakanie servera na neexistujúcu správu a následné uviaznutie. Medusa čakala na odpoveď od autorizačného servera a sever zase dokončenie správy o udalosti, čo spôsobilo nefunkčnosť systému s nutnosťou tvrdého reštartu.

Počas vývoja a ladenia autorizačného servera sa vyskytol jeden zvláštny problém. Nastáva len v prostredí xfce-terminalu so spustenými debug výpismi. Pri štarte ReStabla sa načíta a vykoná prvých pár žiadostí, následne na čo ReStable ukončí svoju činnosť s hlásením *PermissionError: [Errno 1] Operation not permitted*. Po vypnutí debug výpisov však ReStable funguje bez problémov, čo nás vedie k presvedčeniu, že existuje súvislosť medzi nekorektným správaním servera a debug výpismi.

Záver

Cieľom práce bolo zvýšenie bezpečnosti OS Linux. Začiatok práce sprevádza sumár o bezpečnosti, v ktorom boli popísané základné pojmy. Nasledoval prehľad bezpečnosti v Unixe a Linuxe, a ďalej sme rozoberali najpopulárnejšie riešenia dodatočného zabezpečenia OS Linux, ich vlastnosti a princípy fungovania. V druhej časti sa nachádza načrtnutie riešenia Medusa Voyager a autorizačného servera, filozofia tohto projektu a analýza základných konceptov a princípov fungovania riešenia Medusa Voyager. Tretia časť obsahuje samotný popis nového autorizačného servera ReStable, od podrobného návrhu a jeho korekcií, cez implementáciu, až po vyhodnotenie funkčnosti projektu, jeho vplyvu na systém a prípadné problémy implementácie, o ktorých vieme.

Kritickou časťou práce bolo naštudovanie a porozumenie komunikačnému protokolu, ktorý systém Medusa Voyager používa. Bez toho by tento projekt nebolo možné realizovať. Taktiež bolo podstatné naštudovať si rozdiely pri použití rôznych konštruktov v jazyku Python a ich dopad na výkon celého riešenia.

Testovaním autorizačného servera sa nám podarilo overiť funkčnosť a správnosť jeho implementácie. Autorizačný server dokáže bezproblémovo prijímať, spracovať a odpovedať na správy, ktoré mu z jadra preposiela modul.

Naším návrhom na ďalšie smerovanie projektu je rozšírenie autorizačného servera ReStable o rozhodovací modul, čo by bol najväčší krok vpred pre tento projekt. Tým by sa mohla konečne Medusa Voyager dostať do hlavnej vetvy jadra.

Najväčším osobným prínosom z tejto práce pre nás sú nové programátorské skúsenosti a prehĺbenie znalostí o bezpečnosti v Linuxe.

Zoznam použitej literatúry

- [1] KÁČER, J. Medúza DS9. Master's thesis, Slovenská technická univerzita v Bratislave, 2014.
- [2] KRUSE, L. Performance Tips. <https://wiki.python.org/moin/PythonSpeed/PerformanceTips>, apríl 2016.
- [3] LTD., C. AppArmor: About. <http://wiki.apparmor.net/index.php/AppArmor:About>, máj 2016.
- [4] MORRIS, J. Basic Concepts. <https://selinuxproject.org/page/BasicConcepts>, november 2009.
- [5] MORRIS, J. Overview of Linux Kernel Security Features. <https://www.linux.com/learn/overview-linux-kernel-security-features>, júl 2013.
- [6] MOSER, J. R. Grsecurity/Overview. <https://en.wikibooks.org/wiki/Grsecurity/Overview>, november 2015.
- [7] SCHAUFLEER, C. Description from the Linux source tree. http://schaufler-ca.com/description_from_the_linux_source_tree, august 2011.
- [8] TAKEDA, K. TOMOYO Linux Overview. <https://osdn.jp/projects/tomoyo/docs/lca2009-takeda.pdf>, január 2009.
- [9] ZELEM, M. Integrácia rôznych bezpečnostných politík do OS Linux. Master's thesis, Slovenská technická univerzita v Bratislave, 2002.

Prílohy

| | | |
|---|---|----|
| A | Štruktúra elektronického nosiča | II |
|---|---|----|

A Štruktúra elektronického nosiča

```
\
\BP_Nagy_2016_ReStable.pdf
\medusa_log_20_5_2016.csv
\restable
\restable\restable.py
\restable\restable
\restable\restable\__init__.py
\restable\restable\constants.py
\restable\restable\entity.py
\restable\restable\kobject.py
\restable\restable\loader.py
\restable\restable\logger.py
\restable\restable\msg_handler.py
```