SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-86243

PODPORA AUDIT SYSTÉMU PRE BEZPEČNOSTNÝ MODUL MEDUSA

BAKALÁRSKA PRÁCA

2020 **Peter Ňaňko**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-86243

PODPORA AUDIT SYSTÉMU PRE BEZPEČNOSTNÝ MODUL MEDUSA

BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Roderik Ploszek

Bratislava 2020 **Peter Ňaňko**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020
Evidenčné číslo: FEI-5382-86243

ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Peter Ňaňko**
ID študenta: 86243
Študijný program: aplikovaná informatika
Študijný odbor: informatika
Vedúci práce: Ing. Roderik Ploszek

Názov práce: **Podpora audit systému pre bezpečnostný modul Medusa**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Existujúce bezpečnostné moduly v operačnom systéme Linux využívajú audit systém na zaznamenávanie udalostí v systéme súvisiacich s bezpečnosťou. Medusa na zaznamenávanie udalostí využíva autorizačný server v užívateľskom priestore.

Úlohy:
1. Analyzujte použitie audit systému v operačnom systéme Linux
2. Navrhnite typy správ, ktoré sa budú zaznamenávať
3. Implementujte navrhnuté riešenie do modulu Medusa
4. Zhodnoťte prínos implementovanej funkcionality

Zoznam odbornej literatúry:

1. Pikula, M. *Distribuovaný systém na zvýšenie bezpečnosti heterogénnej počítačovej siete.* Diplomová práca. STU, 2002.

Riešenie zadania práce od: 23. 09. 2019
Dátum odovzdania práce: 01. 06. 2020

Peter Ňaňko
študent

Dr. rer. nat. Martin Drozda
vedúci pracoviska

prof. Dr. Ing. Miloš Oravec
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Peter Ňaňko
Bakalárska práca:	Podpora audit systému pre bezpečnostný modul Medusa
Vedúci záverečnej práce:	Ing. Roderik Ploszek
Miesto a rok predloženia práce:	Bratislava 2020

Cieľom tejto práce je implementácia už existujúceho Linux audit systému pre bezpečnostný modul Medusa v operačnom systéme Linux. Modul Medusa na zaznamenávanie udalostí používa autorizačný server, ktorý sa nachádza v užívateľskom priestore. Z toho vyplýva, že pokiaľ prebieha systémová udalosť, ktorá je na bezpečnostnej úrovni kernel modulu Medusy a je zamietnutá z dôvodov architektúry bezpečnostného modulu alebo prázdneho prieniku virtuálnych svetov, tak autorizačný server danú udalosť neauditu je, prečo bola zamietnutá. Linux audit systém ponúka nástroje na prácu s auditom. Preto do Medusy bol aplikovaný audit systém s návrhom správ na auditovanie, ktorý tieto udalostí zaznamenáva, ukladá a zapisuje ich do daného súboru. V mojej práci opisujem použitie audit systému v operačnom systéme Linux a implementáciu audit systému ďalšími majoritnými Linux Security Modulmi. Ďalej popisujem navrhovanie typov správ na zaznamenávanie daných udalostí a ich implementáciu do modulu Medusa. Na konci práce demonštrujem použitie audit systému pre modul Medusa a analyzujem implementovanú funkcionalitu.

Kľúčové slová: Linux, LSM, Medusa, Constable, audit systém

ABSTRACT

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

Study Programme:	Applied Informatics
Author:	Peter Ňaňko
Bachelor's thesis:	Audit System Support for Medusa Security Module
Supervisor:	Ing. Roderik Ploszek
Place and year of submission:	Bratislava 2020

The goal of this work is the implementation of an existing Linux audit system for the Medusa security module in the Linux operating system. The Medusa module uses an authorization server located in the user space to record events. It follows that if a system event occurs that is at the kernel modul Medusa security level and is rejected due to the architecture of the security module or not on the intersection of virtual worlds, the authorization server does not audit why the operation is not permitted. The Linux audit system offers tools for working with audits. Therefore, an audit system was applied to Medusa with a draft of audit reports, which records, saves and writes these events to a given file. In my work I describe the use of audit system in the Linux operating system and the implementation of audit system by other major Linux Security Modules. Next, I describe the design of message types for recording given events and their implementation in the Medusa module. At the end of the work, I demonstrate the use of the audit system for the Medusa module and analyze the implemented functionality.

Keywords: Linux, LSM, Medusa, Constable, audit system

Poďakovanie

Moje poďakovanie patrí vedúcemu tejto bakalárskej práce Ing. Roderikovi Ploszekovi, ktorý ma vždy vedel naviesť na správnu cestu. Za jeho ochotu, čas, trpezlivosť a poznatky k danej téme a problematike.

Obsah

Zoznam obrázkov

Zoznam ukážok kódu a príkazového riadku

Úvod

V práci sa zaoberám podporou audit systému pre Linux Security Modul Medusa. Linux Security Modul Medusa sa skladá z dvoch dôležitých častí, kernel modulu, ktorý implementuje LSM systém na integráciu so systémovými volaniami v Linuxe a autorizačného serveru, ktorý rozhoduje o prístupe a jednotlivé prístupy audituje.

Bezpečnostný prístup v systéme nastáva vtedy ak subjekt (proces) žiada o právo manipulácie s objektom (zapisovať, čítať). LSM Medusa na vyhodnotenie bezpečnostného prístupu používa virtuálne svety a následne posiela žiadosť autorizačnému serveru o povolenie prístupu. V systéme Medusy môže dôjsť k takému bezpečnostnému stavu, kedy sa prístup nepovolí a zamietne sa na úrovni kernel modulu Medusa, kvôli prázdnemu prieniku virtuálnych svetov subjektu a objektu. Autorizačnému serveru sa neposiela správa o zamietnutom prístupe a autorizačný server prístup neauditue.

Linux audit systém ponúka nástroje pre prácu s auditom. Preto do LSM Medusa je aplikovaný Linux audit systém. Linux poskytuje audit systém, ktorý spoľahlivo zaznamenáva informácie o bezpečnostne relevantných udalostiach (aj nerelevantných) v operačnom systéme. V prvých sekciách práce rozoberám Linux Security Module a LSM Medusa. Neskôr sa venujem Linux audit systému, jeho funkcionalite, jeho aplikácii a porovnávam audit ostatných LSM modulov. Ďalej sa venujem implementácií audit systému do LSM Medusa a jednotlivým audit správam. Na záver analyzujem zmysluplnosť aplikovanej funkcionality a možné zlepšenia.

1 Úvod do Linux Security Modulu

LSM je generická softvérová štruktúra, ktorá umožňuje Linux kernelu podporu pre viaceré security moduly, napr. `SELinux, Smack, AppArmor, TOMOYO, Medusa`. Súčasťou Linux kernelu je od verzie `2.6`. Rozhranie sa da nájsť v zdrojovom Linux adresári pod zložkou `/security` [1].

LSM sa primárne zameriava na kontrolu prístupu, hoci jeho vývoj poukazuje aj na ďalšie bezpečnostné potreby, ktorým je napríklad aj audit. Pre systematické opisovanie udalostí, LSM moduly používajú model abstrakcie delenia kernelovských entít na **subjekty** a **objekty**:

Subjekt – je taká kernelovská entita, ktorá vykonáva operáciu (prístup) na objekte. Táto operácia je následne podrobená kontrole prístupu. Subjekty sú skoro stále reprezentované procesom v systéme.

Objekt – je taká kernelovská entita, nad ktorou je operácia (prístup) vykonávaná. V operačnom systéme to môže byť čokoľvek, napr. súbor, proces alebo socket a rôzne ďalšie objekty.

1.1 Postup bezpečnostných udalostí v OS Linux

Bezpečnostné udalosti v operačnom systéme Linux prebiehajú v tomto poradí [2]:

1. **Discretionary Access Control (DAC)** – tvoria ho UNIX kontroly povolení súborov a POSIX Access Control Lists (**ACL**), každý súbor má tri sady privilégií, majiteľa, skupinu a ostatných. Privilégia sú čítanie (read), zapisovanie (write) a spustenie (execute), porovnávajú sa **UID** a **GID** procesu (subjektu) a súboru (objektu).

2. **Capabilities** – Linux tradične delí oprávnenia spojené so super-užívateľom (superuser) do rôznych jednotiek, známych ako capabilities, ktoré môžu byť nezávisle aktivované a de-aktivované pre jednotlivé vlákna.

3. **Minor LSM moduly (LoadPin, Yama)** – obvykle obsahujú príznaky na zapnutie a vypnutie možností na rozdiel od politík súborov, ktoré sú načítané z užívateľského priestoru ako súčasť spustenia systému. Od majoritných modulov sa líšia tým, že ich môže byť viac a nepotrebujú security polia.

4. **Majoritný LSM modul** – môže byť spustený iba jeden počas behu systému, je volaný ako posledný.

1.2 LSM rozhranie

LSM pridáva do dátových štruktúr Linux kernelu security polia a vkladá `security()` volania v kritických častiach kódu, ktoré sú sprevádzané handler funkciami, predtým kedy kernel ide realizovať systémové volanie, vtedy je bezpečnostný modul zavolaný a na základe argumentov a súvislosti udalostí otvoreného systémového volania môže operáciu odmietnuť alebo povoliť.

Aby táto funkcionalita mohla fungovať bezpečnostný modul musí registrovať handler funkcie (tzv. hooks), ktoré sú volané v týchto kritických oblastiach. Ukážka nižšie demonštruje systémové volanie `create`. Volaná funkcia `security_inode_create()` reprezentuje LSM hook, ktorý odovzdá argumenty funkcii bezpečnostného modulu a tá rozhodne o kontrole prístupu. Nižšie je ukážka registrovania hooku funkcie bezpečnostného modulu.

```
1  int vfs_create(struct inode *dir, struct dentry *dentry, umode_t mode,
2                 bool want_excl)
3  {
4          int error = may_create(dir, dentry);
5          if (error)
6                  return error;
7
8          if (!dir->i_op->create)
9                  return -EACCES; /* shouldn't it be ENOSYS? */
10         mode &= S_IALLUGO;
11         mode |= S_IFREG;
12         error = security_inode_create(dir, dentry, mode);
13         if (error)
14                 return error;
15         error = dir->i_op->create(dir, dentry, mode, want_excl);
16         if (!error)
17                 fsnotify_create(dir, dentry);
18         return error;
19 }
```

Príklad 1: Systémového volanie `create` Linux kernelu

```
1                LSM_HOOK_INIT(inode_create, medusa_l1_inode_create)
```

Príklad 2: Registrácia hooku funkcie bezpečnostného modulu

Poznáme dva typy LSM hookov [2]:

Security Hooks – sú to hooky reprezentujúce kontrolu bezpečnostného prístupu. Nachádzajú sa na miestach v kóde, kde subjekty v užívateľskom priestore sú na úrovni prístupu ku kernelovskému objektu. Potom bezpečnostný modul rozhodne, či je prístup povolený alebo zamietnutý.

Control Hooks – tieto hooky nereprezentujú bezpečnostný prístup, ale slúžia na účel oboznámenia bezpečnostného modulu o dôležitých udalostiach v systéme, ktoré by ich mohli zaujímať. Príkladom sú `alloc` a `free` hooky. Tieto hooky sú zavolané pri vytváraní/uvoľňovaní kernelovského objektu bezpečnostným modulom na alokáciu alebo de-alokáciu security polí, ktoré sú v bezpečnostnom module reprezentované ako **security blobs**.

Ďalšou dôležitou súčasťou LSM sú tieto security polia jednoduchého typu `void *` čo znamená, že môžu ukazovať na rôzne dátové štruktúry, v majoritných LSM sú reprezentované ako **security blobs**, sú definované ako ukazovatele na dátove štruktúry, ktoré sú riadené bezpečnostným modulom pre uloženie informácie o systémovom objekte. Príklad kernel dátových štruktúr, ktoré obsahujú tieto security blob ukazovatele sú [3]:

`bpf_map` – key-value štruktúra rozšíreného Barkeley Packet Filtra

`cred` – security kontext procesu

`file` – popisuje otvorený súbor

`inode` – popisuje súbor na filesystémoch

`kern_ipc_perm` – obsahuje zdrojové IPC informácie

`key` – autorizačný token, prihlasovacie údaje, keyring

`msg_msg` – obsahuje dáta o správe

`sock` – reprezentuje socket v sieťovej vrstve

`super_block` – popisuje metadáta súborového systému

`task_struct` – reprezentuje vlákno procesu

2 Medusa Voyager System Security

Medusa je bezpečnostný modul, ktorý je vyvíjaný na Fakulte elektrotechniky a informatiky Slovenskej technickej univerzity v Bratislave od roku 1997. Keďže vývoj modulu začal pred rokom 2004 kedy LSM nebol ešte aplikovaný oficiálne do Linuxového kernelu, tak Medusa DS9 musela byť implementovaná ako kernel záplata.

Podpora a rozvoj na tomto projekte nejakú dobu utíchol, ale opäť sa obnovil v roku 2014 kedy sa kernel modul Medusa premiestnil do LSM na nový 64-bitový kernel. Práce pokračovali na autorizačných serveroch, rozšírení systémových volaní, testovacích prostrediach, kontroly prístupov medziprocesnej komunikácie, konkurencie pre viac systémových volaní a kontroly činností súvisiacich so sieťou.

Architektúra Medusy sa skladá z dvoch dôležitých častí. Kernel LSM Medusa a autorizačného servera, ktorý sa nachádza v užívateľskom prostredí, ktorého sa LSM Medusa pýta o povolenie prístupu (operácie).

2.1 LSM Medusa

Architektúru LSM Medusy je typická svojou vrstvovitou štruktúrou. Tvorí ju aktuálne päť vrstiev (L0–L4), ktoré sú od seba izolované, aby jednotlivé modifikácie vo vrstvách neovplyvňovali vo veľkej miere nasledujúce vrstvy. Predtým ako si vysvetlíme jednotlivé vrstvy, popíšem dôležité štruktúry LSM Medusa [4]:

1. **K-objekt** – táto štruktúra reprezentuje objekt Linux kernelu, kde sú skopírované nutné informácie, dáta ohľadom kontroly prístupu a následne sú posielané autorizačnému serveru.

2. **K-trieda** – je definovaná k-objektom spoločne s operáciami na nasledovnú manipuláciu s daným k-objektom. Operácie sú ukazovatele na jednotlivé funkcie, ktoré tieto operácie vykonávajú. Obsahuje ešte list atribútov k-objektu z ich veľkosťami. Jednotlivé operácie sú:

   - fetch operácia
   - update operácia

3. **Virtuálne svety (VS)** – sú to bitové polia reprezentované ako 32-bitový `int`, čo znamená, že každý bit patrí jednému virtuálnemu svetu (max. 32). Samozrejme užívateľ môže zmeniť veľkosť bit-mapy virtuálnych svetov na jeho zvolené číslo. Zmena virtuálnych svetov má za následok vyžiadanie opätovnej kompilácie Linux kernelu.

4. **Security blob** – obsahuje dáta, ktoré závisia od toho, v ktorej kernelovskej entite alebo štruktúre sa nachádzajú. Napríklad:

   - **Subjekt** – obsahuje tri sady VS[1], list akcií a ukazovateľ na zreťazený zoznam pre registrované k-triedy a udalosti.
   - **Objekt** – obsahuje sadu VS, list akcií, hodnotu magic[2], ukazovateľ na zreťazený zoznam pre registrované k-triedy a udalosti.

   Na základe preseknutí VS medzi subjektom a objektom LSM Medusa definuje tri typy prístupu.

   - **Čítať (read)** – umožňuje subjektu čítať dáta z objektu.
   - **Zapisovať (write)** – umožňuje subjektu zapisovať dáta v objekte.
   - **Vidieť (see)** – umožňuje subjektu vidieť, či objekt existuje.

Keďže LSM Medusa komunikuje s autorizačným serverom v užívateľskom priestore, je aplikovaná vrstvená architektúra, kde nižšie vrstvy obsahujú kernelovské entity, aplikáciu LSM generickej štruktúry a postupne vyššie dochádza ku kopírovaniu dôležitých informácií kernelovských entít na k-objekty a ku komunikácii s autorizačným serverom. Jednotlivé vrstvy tvoria:

L0 vrstva – je zodpovedná za registrovanie control hookov a ich neskoršiu inicializáciu v systéme.

L1 vrstva – je zodpovedná za registráciu security hookov funkciám bezpečnostného modulu a následne volá funkcie z vrstvy L2.

L2 vrstva – Je to hlavná vrstva, ktorá definuje jednotlivé udalosti a typy prístupu, kde prebieha kontrola. Presekávajú sa tu virtuálne svety subjektu a objektu, pričom ak sa VS nepreseknú, operácia prístupu je zamietnutá. Pokiaľ sa VS preseknú a prístup je monitorovaný autorizačným serverom, autorizačná požiadavka je poslaná autorizačnému serveru. V tejto vrstve ešte dochádza k preklápaniu dôležitých údajov kernelovských entít na k-objekty, následne prebieha fetch a update operácia používaná autorizačným serverom pre každý typ k-objektu.

L3 vrstva – Táto vrstva je zodpovedajúca za registráciu k-triedy autorizačným serverom. Definuje dátové typy a pomocné makrá používane Medusou.

[1] každý je nasledovne porovnávaný pri kontrole prístupu

[2] slúži na kontrolu toho, či autorizačný server vie o danom objekte

L4 vrstva – Vrstva poskytuje súbežnú komunikáciu s autorizačným serverom o tom, či bude daná operácia povolená alebo zamietnutá.

2.2 Autorizačný server

Je to užívateľský proces, ktorý rozhoduje o jednotlivých monitorovaných operáciach. Môže byť implementovaný v rôznych jazykoch, avšak autorizačný server musí používať špecifický komunikačný protokol definovaný v Linux kerneli, takže je možné implementovať plnohodnotný autorizačný server iba tým, že programátor pozná tento protokol [4].

Protokol umožňuje komunikáciu vo forme paketov, ktoré nesú všetky potrebné údaje. Komunikácia medzi kernelom a autorizačným serverom prebieha medzi špeciálnym zariadením na komunikáciu `/dev/medusa` [5].

Aktuálne existujú dve implementácie autorizačných serverov:

Constable

- implementovaný v jazyku C
- zjednotený menný priestor ukladá všetky objekty – ukladá informácie o kernel entitách a o objektoch autorizačného serveru
- modulárna architektúra, ktorá separuje funkcionalitu do jednotlivých modulov.

mYstable

- implementovaný v jazyku Python
- jeho funkciou nebolo nahradiť originálny server, ale rýchla implementácia a testovanie nových funkcionalít bezpečnostného modulu.
- nemá implementovaný zjednotený menný priestor ani VS model

Implementácia autorizačného serveru v užívateľskom prostredí má za následok to, že zmeny kernelu sú jednoduchšie, menšie, prechádzanie medzi verziami Linuxu sú flexibilnejšie a zmeny autorizačného serveru nevyžadujú modifikácie Linux kernelu.

3 Linux Audit Systém

Linux audit systém poskytuje auditový subsystém, spĺňajúci požiadavky bezpečnostných osvedčení, dodržiavaním predpisov, certifikácií napr. je kompatibilný s **CAPP (Controlled Access Protection Profile)** [6]. Spoľahlivo zaznamenáva informácie o akejkoľvek bezpečnostne relevantnej udalosti (alebo aj nerelevantnej) v systéme. Tento audit systém pomáha sledovať udalosti, ktoré sú vykonávané v systéme.

Úlohou auditu je zvýšiť bezpečnosť systému tým, že poskytuje komponenty na podrobnú analýzu toho, čo sa v danom systéme deje. Neposkytuje však samo osebe rozširujúcu bezpečnosť, nechráni daný systém pred chybnými funkciami kódu ani pred akýmkoľvek druhom zneužitia. Namiesto toho je audit užitočný na sledovanie a monitorovanie týchto udalostí, ktoré pomáha do budúcna riešiť a ďalej podnikať bezpečnostné opatrenia, aby sa takýmto druhom zneužitia alebo nedbanlivosti dalo zabrániť.

Zaznamenané udalosti by mali obsahovať dostatok informácií, aby mohol bezpečnostný analytik (systémový administrátor) zistiť, čo sa stalo v systéme neskôr. Môže ísť napríklad o pravidlá, pri ktorých je prístup k súborom v konkrétnom adresári veľmi dôležitý. Systémový administrátor by napr. obmedzil prístup a umožnil zamýšľaný prístup prostredníctvom **group permission** (skupinových povolení) alebo umiestnením **POSIX ACL**. Ako by však vedel, či zamýšľané bezpečnostné pravidlá fungujú? Pomocou systémového auditu, ktorý je možné nakonfigurovať tak, aby sledoval, kto pristupuje k týmto súborom, a poskytoval informácie ohľadom udalostí. Systémový administrátor potom môže pravidelne kontrolovať audit logy (správy), aby sa ubezpečil, že nedošlo k žiadnemu neautorizovanému prístupu.

3.1 Úvod do audit systému

Linux audit subsystém bol prvýkrát zavedený do linuxového jadra vo verzii `2.6`. Jeho cieľom je sledovať kritické systémové udalosti súvisiace s bezpečnosťou. Audit mapuje procesy k užívateľom, ktorí ich začali. To umožňuje systémovým administrátorom sledovať, ktorý užívateľ vlastní daný proces a tak vyhodnotiť škodlivú udalosť. Tieto druhy udalostí sú nasledujúce[3]:

- neúspešné pokusy o autorizáciu (prihlásenie)
- čítanie, zápis a zmena prístupových práv k súboru
- inicializácia sieťového pripojenia

[3] tento zoznam nie je ani zďaleka úplný, sú uvedené iba niektoré z nich

- zisťovanie chýb systému, napr. padajúce procesy
- zaznamenanie príkazov jednotlivými užívateľmi
- zmena informácií o používateľovi a skupine
- vykonávanie systémových volaní

Predchádzajúce prípady by nemohli nastať bez systémových volaní v kerneli, ktoré zachytí kernel audit modul, pomocou určených pravidiel ich sformuje a zapíše do daného súboru pre audit logy.

3.2 Úloha Linux audit systému

Úlohou Linux audit systému je:

1. Zhromažďovať a zaznamenávať dané udalosti vyskytujúce sa v systéme
   - udalosti Linuxového kernelu (syscall udalosti)
   - udalosti užívateľa
2. Formovať a zapisovať každú udalosť dostatočne jasne pre administrátora do súboru. Napr. `/var/log/audit/audit.log`
3. Ponúknuť jednotlivé komponenty na pomáhanie pri práci s audit logmi. (Napr. vyhľadávanie pri veľkom obsahu logov, nastavovanie pravidiel pre zaznamenávané udalosti)

3.3 Komponenty

Celý Linux audit systém tvorí niekoľko komponentov, z rôznou funkcionalitou, niektoré si predstavíme v nasledovnej sekcii.

3.3.1 Zoznam komponentov

Kernel:

- `audit`: hooky do Linux kernelu, na zachytenie udalostí a ich doručenie `auditd`.

Spustiteľné binárne súbory:

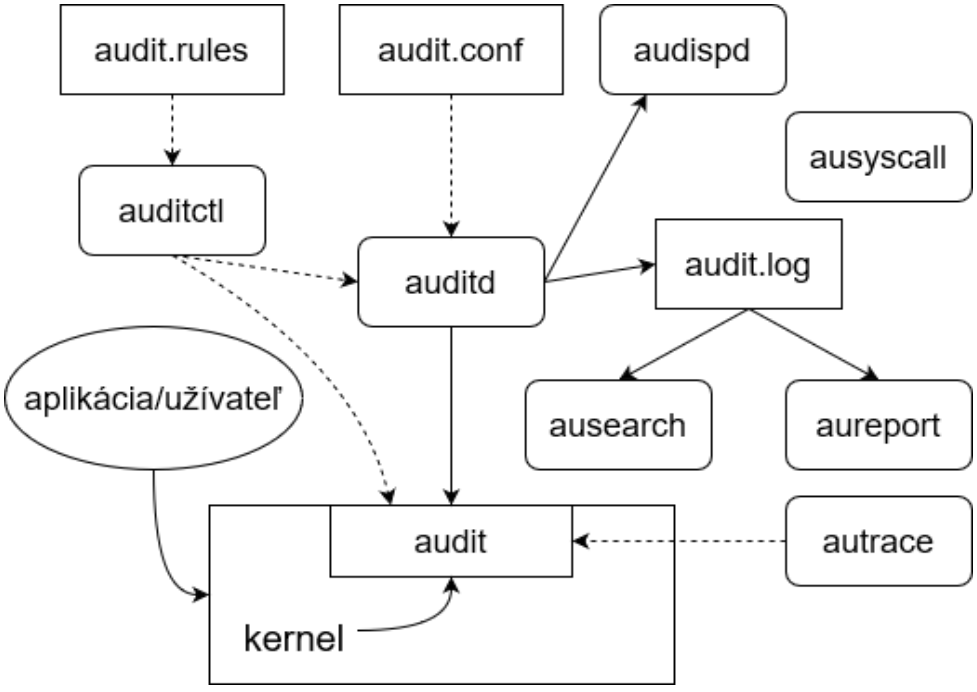
- `audispd`: multiplexor udalostí.
- `auditctl`: klientský nástroj, ktorý pomáha riadiť kernelovský audit systém a pravidlá pre `auditd`.

- `auditd`: užívateľský daemon, slúži na zachytávanie udalostí a zapisovanie do `audit.log`.
- `aureport`: oznamovací nástroj, vytvára sumarizáciu správ z `audit.log`.
- `ausearch`: nástroj na vyhľadávanie udalostí pomocou dotazu z `audit.log`
- `ausyscall`: program, ktorý umožňuje mapovanie syscall mien a čísel.
- `autrace`: používa komponent kernelu audit systému na sledovanie binárnych súborov

Súbory:

- `audit.conf`: konfiguračný súbor pre `auditd`.
- `audit.log`: zoznam všetkých auditovaných udalostí.
- `audit.rules`: súbor pravidiel načítaných v kerneli audit systému.

3.3.2 Diagram závislostí komponentov



Obr. 1: Komponenty audit systému

3.4 Inštalácia

Na spustenie auditu je potrebné stiahnuť a nainštalovať audit deamon a spustiť tento servis. Pre prispôsobený kernel by mal `CONFIG_AUDIT` byť povolený. Audit je možné povoliť aj počas spúšťania systému nastavením `audit=1` ako parameter kernelu.

4 Audit systém v LSM

Keďže audit systém je dôležitou súčasťou bezpečnostného modulu, majoritné moduly `SELinux, Smack, AppArmor, TOMOYO` audit aplikujú.

Kernel ponúka pomocné generické funkcie pre logovanie jednotlivých udalostí audit systémom. Implementácia audit podpory sa nachádza v zdrojových súboroch Linux adresára `/include/linux/audit.h` a `/kernel/audit.c` [1]. Ďalšia podpora spoločného auditu sa nachádza v `/include/linux/lsm_audit.h` a `/security/lsm_audit.c` [1]. Jednotlivé funkcionality podpory auditu využívajú `SELinux, Smack` a `AppArmor` pri implementovaní Linux audit systému. LSM modul `TOMOYO` Linux audit systém neimplementuje, aplikuje vlastný audit deamon a typy správ.

V nasledujúcich sekciách si predstavíme spoločne naprogramovanú podporu `audit, lsm_audit`, ktorú tieto LSM moduly implementujú. Bližšie si opíšeme ukážky kódu, tok informácií, generické funkcie a štruktúry využívané pri implementácií a návrhu auditu.

4.1 Definovanie dát pre audit

Na to aby sa zapísali dôležite dáta pre audit musia byť niekde uložené. Majoritné moduly ukladajú tieto dáta v dátovej štruktúre `common_audit_data`. Pri implementácií auditu modulu Medusy sa ukazovateľ na jej audit dáta uloží do tohto unionu spoločne s konfiguračnou podmienkou `CONFIG_SECURITY_MEDUSA`.

- Ukazovateľ na konkrétne dáta bezpečnostného modulu.

```
1  /* this union contains LSM specific data */
2  union {
3  #ifdef CONFIG_SECURITY_SMACK
4            struct smack_audit_data *smack_audit_data;
5  #endif
6  #ifdef CONFIG_SECURITY_SELINUX
7            struct selinux_audit_data *selinux_audit_data;
8  #endif
9  #ifdef CONFIG_SECURITY_APPARMOR
10           struct apparmor_audit_data *apparmor_audit_data;
11 #endif
12 }; /* per LSM data pointer union */
```

Príklad 3: Audit dáta

- Indikátory definujúce typ objektu, ktorý bude auditovaný, obsah tejto štruktúry je samozrejme oveľa väčší, bola skrátená iba pre uvedenie príkladu.

```
char type;
#define LSM_AUDIT_DATA_PATH   1
#define LSM_AUDIT_DATA_NET    2
#define LSM_AUDIT_DATA_CAP    3
#define LSM_AUDIT_DATA_IPC    4
#define LSM_AUDIT_DATA_TASK   5
#define LSM_AUDIT_DATA_INODE  9
#define LSM_AUDIT_DATA_DENTRY 10
union {
          struct path path;
          struct dentry *dentry;
          struct inode *inode;
          struct lsm_network_audit *net;
          int cap;
          int ipc_id;
          struct task_struct *tsk;
} u;
```

Príklad 4: Indikátory a typ objektu

Na týchto implementáciách je zaujímavé kľúčové slovo `union` - umožňuje ukladať rôzne typy údajov na rovnaké miesto v pamäti. Môže definovať viaceré členy, ale iba jeden člen môže obsahovať hodnotu v ľubovoľnom čase. `union` poskytuje efektívny spôsob použitia rovnakého umiestnenia pamäte na viaceré účely a tvorenia generickosti.

4.2 Formátovanie a zápis audit správ

Pomocná generická funkcia `common_lsm_audit` umožňuje, aby sa dané informácie uložili a následne zapísali. Je to generická audit funkcia, jej vstupnými parametrami je už opísaná štruktúra `common_audit_data` a dva ukazovatele `void *` na LSM špecifickú `pre_audit` a `post_audit` funkciu. Ukazovatele na tieto dve auditové funkcie definuje a posiela konkrétne používaný LSM modul, v ktorých formátuje audit informácie do audit správy podľa svojich predstáv a potrieb oboznámenia, čo sa v LSM deje. Špecifické funkcie pre jednotlivé moduly si opíšeme v ďalších sekciách.

```
void common_lsm_audit(struct common_audit_data *a,
          void (*pre_audit)(struct audit_buffer *, void *),
          void (*post_audit)(struct audit_buffer *, void *))
{
          struct audit_buffer *ab;

          if (a == NULL)
                    return;
          /* we use GFP_ATOMIC so we won't sleep */
          ab = audit_log_start(audit_context(), GFP_ATOMIC | __GFP_NOWARN,
                                    AUDIT_AVC);

          if (ab == NULL)
                    return;

          if (pre_audit)
                    pre_audit(ab, a);

          dump_common_audit_data(ab, a);

          if (post_audit)
                    post_audit(ab, a);

          audit_log_end(ab);
}
```

Príklad 5: Audit dát generickou funkciou

Jednotlivé audit informácie sú vkladané do audit buffra (`ab`), ktoré sú väčšinou formátované ako `key=value` páry. Táto funkcia vykonáva nasledujúci sled udalostí:

1. Funkcia `audit_log_start` inicializuje `ab`, naplní ho audit typom (`type=AVC`) [7] a aktuálnym časom (`msg=audit(1588558801.69:184)` unixového timestampu. Príklad 12 vysvetľuje konvertovanie hodnoty timestampu na čitateľný a zrozumiteľný dátum a čas.

2. LSM modul uloží vlastnú audit správu do `ab` v `pre_audit` funkcii, ktorá je následne volaná.

3. Vo funkcii `dump_common_audit_data` sa uložia informácie do `ab` o konkrétnom procese (subjekte), jeho `pid` [7] (proces ID) a `comm` [7] (meno spusteného príkazu), neskôr sa uložia informácie o danej entite (objekte), ktorý inicializuje LSM pomocou indikátorov a kópie entity, nad ktorou je operácia vykonávaná.

4. LSM modul uloží vlastnú audit správu do `ab` v `post_audit` funkcii, ktorá je následne volaná.

5. Nakoniec funkcia `audit_log_end` prekopíruje `ab` do socket buffra, ktorý prenáša správu audit deamonu (`auditd`) a zapisuje do audit logu (`audit.log`), následne je `ab` uvoľnený.

4.3 Detailné formátovanie

Pre detailné formátovanie správ v `pre_audit` a `post_audit` funkciách o konkrétnom LSM module, sú implementované pomocné funkcie. Slúžia pre konkrétne, bezpečné a ošetrené uloženie parametru/ov do audit buffra (`ab`). Zoznam a popis niektorých z funkcií:

- `audit_log_format` – funkcia formátuje správu do `ab`. Ak formát reťazca obsahuje špecifikátor formátovania (subsekvencie začínajúce `%`), ďalšie argumenty funkcie nasledujúce po formáte reťazca, sa naformátujú a vložia do výsledného reťazca, ktorý nahradí ich príslušné špecifikátory.

- `audit_log_string` – funkcia vloží formátovaný predpokladane bezpečný charakterový reťazec do `ab`.

- `audit_log_untrustedstring` – funkcia zaloguje do `ab` reťazec, ktorý môže obsahovať náhodné znaky. Ak reťazec obsahuje kontrolný znak, nevypisateľný znak, znak úvodzoviek alebo medzeru, funkcia sa im vyhne. Reťazce/znaky, ktorým sa funkcia vyhla sú vypísané v hexadecimálnej číselnej sústave (dve číslice na znak). Táto funkcia sa väčšinou používa pri vstupoch názvov, ktoré zadával užívateľ.

5 Porovnanie auditu u LSM modulov

Informácie k jednotlivým audit správam v LSM moduloch sú aktuálne k verzii Linux kernelu `v5.5`.

5.1 SELinux a bezpečnostná politika

Keď subjekt žiada o prístup k objektu, `SELinux` skontroluje access vector cache (`AVC`), kde sa rozhodnutia ukladajú do pamäte cache pre subjekty a objekty. Ak `SELinux` nedokáže rozhodnúť o prístupe na základe oprávnení uložených v pamäti, pošle žiadosť bezpečnostnému serveru (`AVC` robí rozhodovanie rýchlejšie ako bezpečnostný server, `AVC` ukladá rozhodnutia servera). Bezpečnostný server skontroluje bezpečnostný kontext subjektu a objektu. Kontext zabezpečenia sa používa z databázy kontroly prístupu. Povolenie vykonať operáciu sa potom udelí alebo zamietne.

`SELinux` pre vyhodnotenie prístupu používa označenia pre subjekty a objekty nazývané `context (user:role:type:level)` [8], ktoré sú definované takto:

- **SELinux user** - používatelia systému Linux sú mapovaní na používateľov `SELinuxu` (nie sú to isté), používatelia definujú, do ktorých rolí má používateľ povolené ísť.
- **role** - roly určujú, do ktorých domén má používateľ povolené ísť.
- **type** - definuje domény, najdôležitejšia časť bezpečnostného kontextu.
- **level** - nepovinná informácia, definuje `sensitivity level` a `category` v používanom `multi-level` bezpečnostnom modely.

5.2 Typy správ auditu SELinux-u

`SELinux` používa šikovné mapovanie objektu triedy (`tclass`), konkrétne jej meno (`name`) voči poľu povolení (`perms`), špecifickej triedy objektu. Implementácia sa dá nájsť v `/security/selinux/include/avc_ss.h`, `/security/selinux/include/classmap.h` [1]

5.2.1 SELinux `pre_audit`

- `avc:` špecifikuje prístup, môže byť `denied` alebo `granted` (zamietnutý/povolený)
- `{ permission }` špecifikuje súbor požadovaných povolení triedou objektu (`tclass`), ktoré boli počas danej operácie. Napr. `{ read }` (čítať).

5.2.2 SELinux `post_audit`

- a) `ssid=` bezpečnostný číselný identifikátor subjektu, je vypísaný ak sa nepodarí dostať `scontext`.

b) `scontext=` bezpečnostný kontext subjektu.

- a) `tsid=` bezpečnostný číselný identifikátor objektu, je vypísaný ak sa nepodarí dostať `tcontext`.

  b) `tcontext=` bezpečnostný kontext objektu (target).

- `tclass=` meno triedy objektu, (napr. file, dir)
- `permissive=` prípustný mód, `0` alebo `1`.
- Pri splnení popísaných podmienok sa vypíšu:

  `srawcon=` v prípade neplatného `scontext`-u sa uvedie aktuálny reťazec `scontext`.

  `trawcon=` v prípade neplatného `tcontext`-u sa uvedie aktuálny reťazec `tcontext`.

5.3 Smack a bezpečnostná politika

`Smack` používa label systém. Bezpečnostné label subjektu a objektu sú reprezentované ako reťazce, ktoré sú následne vyhodnocované pravidlami, či má subjekt prístup k objektu danými prístupmi. `Smack` má 7 druhov prístupov: `read(r), write(w), execute(x), append(a), transmute(t), lock(l), bringup(b)`. Prvých šesť prístupov je bezpečnostných. Bringup [9] je pre vyvojárov a znamená, že všetky úspešné prístupy budú v systéme zaznamenané s aktuálnymi právami, ktoré boli potrebné na vykonanie operácie (tento režim bol vytvorený na pomoc pri vytváraní pravidiel pre vývojárov, pretože všetky zaznamenané prístupy je možné analyzovať). Smack podporuje pravidlá na zaznamenávanie pri audite.

5.4 Typy správ auditu Smack-u

`Smack` používa pri inicializácií audit štruktúr inline funkcie (čím sa ušetrí réžia volania funkcie), ktoré audit urýchľujú. Niektoré inline funkcie na nastavenie audit údajov nerobia nič, pokiaľ nie je nastavený `CONFIG_AUDIT`.

5.4.1 Smack `pre_audit`

- `fn=` meno funkcie daného prístupu.
- `action=` špecifikuje prístup, môže byť `denied` alebo `granted` (zamietnutý/povolený)
- `subject=` subjekt label reťazec
- `object=` objekt label reťazec

- Vypíše jednu z možností:

  a) `requested=` požadované práva prístupu, zapísané písmeno prístupu. Napr. `requested=r`

  b) `labels_differ` ak sa práva prístupu nezhodovali

5.4.2 Smack `post_audit`

`Smack` nedefinuje `post_audit` funkciu, posiela parameter `NULL`.

5.5 AppArmor a bezpečnostná politika

`AppArmor` aplikuje súbor pravidiel, privilégií (známe ako „`profile`“). Aplikovaný profil závisí od inštalačnej cesty vykonávaného programu. `AppArmor` riadi iba procesy s priradeným profilom. Procesy bez profilu prebiehajú bez obmedzenia, to je bez rozhodovania o prístupe **MAC (Mandatory Access Control)**. `AppArmor` poskytuje aj `aa_audit` mód, ktorý nastaví profil do audit módu, kde sú všetky operácie zaznamenané. Zaznamenávanie operácií je závislé od daného profil módu a nastavených modifikátory pravidiel. Modifikátory pravidiel pre audit sú [10]:

1. `audit` – donúti systém k zaznamenávaniu udalostí
2. `deny` – výlučne odmietne, bez zaznamenávania udalostí
3. `audit deny` – kombinácia s `deny`, ale na rozdiel od neho správu zaznamená

5.6 Typy správ auditu AppArmor-u

`AppArmor` používa makrá pre skrátený, prehľadný a rýchly zápis jednotlivých štruktúr a unionov. Funkcia `pre_audit` je spoločná pre každú audit správu. Funkcia `post_audit` je závislá od druhu prístupu, ktorý opisuje.

5.6.1 AppArmor `pre_audit`

Keďže audit `AppArmoru` je zložitý a veľmi závislý od stavu operácie, nasledovné parametre sú vypísané podľa daného stavu a nemusia byť vypísané všetky.

- `apparmor=` apparmor audit typ, napr. (DENIED)
- `operation=` vykonávaná operácia, napr. (mknod)
- `info=` bližšia informácia o operácii, napr. (Failed name lookup)
  `error=` číselná informácia o probléme pri operácií
- vypíše meno zavedenej konfigurácie a bezpečnostnej politiky profilu.

1. `namespace=` namespace
2. `profile=` profil
3. `label=` label

- `name=` meno danej `path` (cesty).

5.6.2 AppArmor `post_audit`

Zoznam jednotlivých funkcií, ktoré opisujú bližšie informácie[4] o operáciách v systéme:

- audit špecifických informácií `capabilities`.
- audit `file` (súborov) informácií.
- audit konkrétnych `ptrace` polí.
- audit špecifických informácií ohľadom operácie `mount`.
- audit pre špecifickú oblasť `net` (network socket layer).
- audit zmeny bezpečnostných politík.
- audit bezpečnostných politík profilu – rozbalenia, načítania, nahradenia, odstránenia.
- audit bezpečnostnej politiky nastavení `resource` a jeho limít.

[4]dáta nemusia byť vypísané všetky, záleží to od stavu operácie podobne ako pri `pre_audit` funkcii

6 Implementácia auditu do LSM Medusa

Pri implementácií auditu do LSM Medusa si treba uvedomiť dve veci:

- Do ktorej vrstvy Medusy bude audit implementovaný ?
- Aké informácie sa budú auditovať ?

6.1 Výber vrstvy

Najvhodnejšia vrstva pre implementáciu auditu je druhá vrstva, pretože v nej dochádza k preťatiu virtuálnych svetov, kontroly monitorovania prístupu, kopírovaniu dôležitých informácií kernelovských entít na k-objekty a následne odosielanie požiadavky autorizačnému serveru a odpoveď.

6.2 Návrh správ

Audit správa má byť navrhnutá tak, aby bola schopná odpovedať na otázku „kto urobil, komu (subjekt-objektu) a čo bol výsledok" [11].

1. meno funkcie daného prístupu.
2. odpoveď Medusy k danému prístupu.
3. identifikátory definujúce virtuálne svety pri operácií a ich preťatie.
4. identifikátory definujúce, či prístup je monitorovaný autorizačným serverom (vďaka tomu môžeme aj uvažovať o tom, či operácia je odoslaná autorizačnému serveru).
5. virtuálne svety – inicializované iba pre výpis keď prístup bol zamietnutý kvôli nim, slúžia na vypísanie detailov o neautorizovanom prístupe. Použité virtuálne svety závisia od typu prístupu, preto sú zapísané v unione.
6. parametre systémového volania, ktoré sú posielanie aj k-objektom. Sú závislé od typu systémového volania, preto sú zapísané v unione.

```
struct medusa_audit_data {
        const char *function;
        medusa_answer_t med_answer;
        char event;
        char vsi;
        union {
                struct swvs sw;
                struct srwvs srw;
        } vs;
        union {
                const char *name;
                int mode;
                struct {
                        dev_t dev;
                        int mode;
                } mknod;
        } pacb;
}
```

Príklad 6: Skrátený príklad audit dát Medusy

```
        struct common_audit_data cad;
        struct medusa_audit_data mad = { .vsi = VS_SW_N };
audit:
#ifdef CONFIG_AUDIT
        cad.type = LSM_AUDIT_DATA_DENTRY;
        cad.u.dentry = dentry;
        mad.function = __func__;
        mad.med_answer = retval;
        mad.pacb.mknod.dev = dev;
        mad.pacb.mknod.mode = mode;
        cad.medusa_audit_data = &mad;
        medusa_audit_log_callback(&cad, medusa_mknod_pacb);
#endif
```

Príklad 7: Skrátený príklad inicializácie dát a volania auditu

6.3 Medusa `pre_audit`

- `op=` meno vykonávanej operácie.
- `ans=` odpoveď Medusy.
- `vs={ }` oboznamuje o preseknutí virtuálnych svetov.
- `access=` hovorí o tom, či je prístup monitorovaný alebo nedefinovaný, napr. pri preseknutí virtuálnych svetov.

6.4 Medusa `post_audit`

Závisí od typu operácie, vypíše bližšie informácie o parametroch, ktoré sú súčasťou k-objektu a posielané autorizačnému serveru.

Zoznam operácií volaní s `post_audit`-om, ktoré sa nachádzajú v L2 vrstve:

- `medusa_create`
- `medusa_ipc_associate`
- `medusa_ipc_ctl`
- `medusa_ipc_msgrcv`
- `medusa_ipc_msgsnd`
- `medusa_ipc_permission`
- `medusa_ipc_semop`
- `medusa_ipc_shmat`
- `medusa_link`
- `medusa_mknod`
- `medusa_permission`
- `medusa_rename`
- `medusa_rmdir`
- `medusa_symlink`

6.5 Implementačné rozdiely

```
user: /linux-medusa$ git diff --shortstat master
23 files changed, 785 insertions (+), 105 deletions (-)
```

Príklad 8: Modifikácie a rozdiely pri implementácií

Pri implementácií audit systému, distribuovaný systém na správu verzií (`git`) priblížil, že vo vedľajšej aplikačnej vetve (`audit_system`), voči hlavnej vetve (`master`) bolo modifikovaných 23 súborov, 785 pridaných riadkov a 105 odobraných riadkov.

6.6 Audit logy

Audit správa (Príklad 9) nám ponúka informácie o type a čase prístupu, o operácií Medusy (`medusa_rmdir`), jej odpoveď na danú udalosť (`ALLOW`), oboznamuje o preseknutí virtuálnych svetov (`intersect`), o tom či je prístup monitorovaný autorizačným serverom (`MONITORED`), informáciu o subjekte teda identifikačné číslo procesu a meno vykonávaného príkazu v príkazovom riadku (`1845, rmdir`), informácie o objekte konkrétne cestu k danému objektu (`/home/user`), meno zariadenia, ktoré obsahuje daný súbor alebo adresár zaznamenaný v správe (`sda1`), `inode` číslo súboru alebo adresára (`393219`) a meno konkrétneho súboru alebo adresára (`testfile`).

```
type=AVC msg=audit(1590681224.164:7877): Medusa: op=medusa_rmdir ans="ALLOW"
    vs={ intersect } access=MONITORED pid=1845 comm="rmdir" path="/home/user"
    dev="sda1" ino=393219 name="testfile"
```

Príklad 9: Správa audit logu `path` hooku

Príklad 10 opisuje zamietnutý prístup, pretože sa virtuálne svety nepresekli.

```
type=AVC msg=audit(1590780362.491:13736): Medusa: op=medusa_permission
    ans="DENY" vs={ see_n , read_n , write_n } access=UNDEFINED pid=2013
    comm="bash" name="work" dev="sda1" ino=263025 mask=1
```

Príklad 10: Správa audit logu `inode` hooku

Audit správy `path` hookov napr. `medusa_rmdir` (Príklad 9) a `medusa_rename` obsahujú cestu k danému súboru alebo adresáru. Medusa zatiaľ pre `inode` hooky `medusa_create`, `medusa_link`, `medusa_symlink`, `medusa_mknod`, `medusa_readlink` a `medusa_permission` (Príklad 10) nepodporuje výpis cesty (`path`). Samozrejme cesta k súboru alebo adresáru sa dá nájsť príkazom v príkazovom riadku, vyhľadaním pomocou `ino` čísla (Príklad 11). Ďalej je uvedené konvertovanie unixového timestampu na nám zrozumiteľný a čitateľný dátum s časom (Príklad 12).

```
user: $ find / -inum 263025 -print
/home
```

Príklad 11: Nájdenie cesty k súboru alebo adresáru pomocou `ino` čísla

```
user: $ date -d @1590780362.491
Fri 29 May 2020 09:26:02 PM CEST
```

Príklad 12: Konvertovanie UNIX timestampu na zrozumiteľný dátum a čas

Záver

Audit systém pomáha zvýšiť bezpečnosť systému tým, že poskytuje prostriedky na podrobnú analýzu toho, čo sa v systéme deje. Neposkytuje však ďalšiu bezpečnosť a nechráni systém pred chybnými funkciami kódu ani akýmkoľvek druhom zneužitia. Namiesto toho je audit užitočný na sledovanie týchto udalostí a pomáha navrhnúť ďalšie bezpečnostné opatrenia, aby sa im zabránilo.

V prvých sekciách sme si vysvetlili ako bezpečnostný modul Medusa funguje v kerneli a vyhodnocuje bezpečnostný prístup pomocou autorizačného serveru. Ďalej sme si predstavili Linux audit systém, jeho nástroje a pomocnú funkcionalitu pri LSM moduloch. Porovnali sme auditované správy s ostatnými modulmi. Následne sme implementovali funkcionalitu a otestovali ju. Audit má aj sekundárnu funkcionalitu, ktorou môže patrične slúžiť na ladenie systému, oboznamovať vyvojárov o danej funkcionalite a stave systému. To audit v autorizačnom servery neponúkal, keďže sa nachádza v užívateľskom priestore oproti Linux audit systému, ktorý sa nachádza v kerneli. Teraz sa autorizačný server môže venovať iba jednej funkcii a to rozhodovaniu o bezpečnostnom prístupe.

Medusa momentálne audituje každý druh prístupu jej zaháknutých funkcií, pre ktoré bol audit aplikovaný. Zatiaľ sme nedošli k žiadnemu implementačnému problému, no to sa po dlhšom čase testovania systému môže zmeniť. Inicializácia dát auditu by sa ešte dala vylepšiť makrami a inline funkciami, ktoré to môžu urýchliť, keďže audit systém o niečo spomaľuje rýchlosť operačného systému. Sekcia auditu v LSM Medusa touto prácou samozrejme nekončí, pretože modul sa môže rozširovať ďalej o funkcionalitu, ktorú bude potreba auditovať. Modul ešte nemá aplikované a zaháknuté všetky pravidlá pre audit, pravidlá zaznamenávania udalostí a cielené funkcie, ktoré budú tento modul do budúcnosti charakterizovať.

Zoznam použitej literatúry

1. *Elixir Cross Referencer: linux* [online]. Bootlin [cit. 2020-05-25]. Dostupné z: `https://elixir.bootlin.com/linux/v5.5/source`.
2. PLOSZEK, R. Linux security modules overview. In: KOZÁKOVÁ, A. (ed.). *ELITECH´18*. Bratislava: Vydavateľstvo Spektrum STU, 2018, ISBN 978-80-227-4794-3.
3. WRIGHT, C., COWAN, C., MORRIS, J., SMALLEY, S., KROAH-HARTMAN, G. Linux Security Modules: General Security Support for the Linux Kernel. *USENIX Security Symposium*. 2002.
4. PIKULA, M. *Distribuovaný systém na zvýšenie bezpecnosti heterogénnej pocítacovej siete*. Master's thesis, Fakulta elektrotechniky a informatiky, 2002.
5. *linux-medusa: Architecture* [online]. Medusa-Team [cit. 2020-05-25]. Dostupné z: `https://github.com/Medusa-Team/linux-medusa#architecture`.
6. JAHODA, M., KRÁTKY, R., PRPIČ, M., ČAPEK, T., WADELEY, S., RUSEVA, Y. a SVOBODA, M. Security Guide: System Auditing. *Red Hat Enterprise Linux 6* [online] [cit. 2020-05-16]. Dostupné z: `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing`.
7. JAHODA, M., KRÁTKY, R., PRPIČ, M., ČAPEK, T., WADELEY, S., RUSEVA, Y. a SVOBODA, M. Security Guide: Appendix B. Audit System Reference. *Red Hat Enterprise Linux 6* [online] [cit. 2020-05-16]. Dostupné z: `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/app-audit_reference`.
8. *SELinux/Type enforcement: Domains* [online] [cit. 2020-05-18]. Dostupné z: `https://wiki.gentoo.org/wiki/SELinux/Type_enforcement`.
9. *Security/Tizen Smack: Smack* [online] [cit. 2020-05-18]. Dostupné z: `https://wiki.tizen.org/Security/Tizen_Smack`.
10. BEATIE, Steve (ed.). *A quick guide to AppArmor profile Language: Rule Modifiers* [online]. AppArmor [cit. 2020-05-18]. Dostupné z: `https://gitlab.com/apparmor/apparmor/-/wikis/QuickProfileLanguage#rule-modifiers`.

11. GRUBB, Steve (ed.). *audit-documentation: SPEC Writing Good Events* [online]. The Linux Audit Project [cit. 2020-05-19]. Dostupné z: `https://github.com/linux-audit/audit-documentation/wiki/SPEC-Writing-Good-Events`.

Prílohy

A Detajlný popis audit správ Medusy

V tejto sekcii si vymenujeme podrobne informácie o audit správach, ktoré sa môžu vyskytnúť pri auditovaní LSM Medusy. Popis je rozdelený na sekcie funkcií, ktoré sú zodpovedné za ukladanie jednotlivých údajov do audit buffra.

A.1 pre_audit

`op=` operácia modulu Medusa

- `medusa_create`
- `medusa_ipc_associate`
- `medusa_ipc_ctl`
- `medusa_ipc_msgrcv`
- `medusa_ipc_msgsnd`
- `medusa_ipc_permission`
- `medusa_ipc_semop`
- `medusa_ipc_shmat`
- `medusa_link`
- `medusa_mknod`
- `medusa_permission`
- `medusa_readlink`
- `medusa_rename`
- `medusa_rmdir`
- `medusa_symlink`

`ans=` odpoveď modulu Medusa

- `ERROR` – error
- `FORCE_ALLOW` – táto odpoveď je v module označená ako zastaralá
- `DENY` – zakáže operáciu
- `FAKE_ALLOW` – táto odpoveď je v module označená ako zastaralá
- `ALLOW` – povolí operáciu

`vs={ }` tento údaj je vypísaný, ak virtuálne svety boli súčasťou operácie

- `intersect` – virtuálne svety sa presekli
- `see_i , read_n , write_n` – typy prístupu[1], hovoria o nepreseknutí virtuálnych svetov
  `_i` – prípona pri type prístupu, znamená preťatie
  `_n` – prípona pri type prístupu, indikuje prázdny prienik svetov

[1] použité typy prístupu závisia od typu operácie

`access=`

- `MONITORED` – prístup je monitorovaný autorizačným serverom
- `UNMONITORED` – prístup nieje monitorovaný autorizačným serverom
- `UNDEFINED` – pri prístupe nedošlo ku kontrole monitorovania

A.2 dump_common_audit_data

`pid=` ID procesu

`comm=` zaznamenaný príkaz[2], ktorý bol spustený

Záleží od typu hooku pri prístupe:

– `path`:

- `path=` cesta k adresáru alebo súboru
- `dev=` meno zariadenia, nachádzajúce sa v `/dev`
- `ino=` číslo `inode`, priradené k adresáru alebo súboru

– `inode`:

- `name=` meno adresáru alebo súboru
- `dev=` meno zariadenia, nachádzajúce sa v `/dev`
- `ino=` číslo `inode`, priradené k adresáru alebo súboru

– `ipc`:

- `key=` IPC ID.

A.3 post_audit

Nasledovné informácie nemusia byť vypísané všetky, informácie závisia od typu operácie:

1. `medusa_create`
   - `mode=` file mód nového súboru
2. `medusa_ipc_associate`
   - `flag=` príznaky operácie

[2]alebo shell interpreter, skript

- `ipc_class=` číselná reprezentácia typu IPC triedy

3. `medusa_ipc_ctl`
    - `cmd=` obsahuje operáciu, ktorá sa má vykonať
    - `ipc_class=` číselná reprezentácia typu IPC triedy
4. `medusa_ipc_msgrcv`
    - `flag=` príznaky operácie
    - `m_type=` typ správy
    - `m_ts=` velkosť textu správy
    - `rm_type=` typ vyžiadanej správy
    - `opid=` proces ID objektu
    - `ipc_class=` číselná reprezentácia typu IPC triedy
5. `medusa_ipc_msgsnd`
    - `flag=` príznaky operácie
    - `m_type=` typ správy
    - `m_ts=` velkosť textu správy
    - `ipc_class=` číselná reprezentácia typu IPC triedy
6. `medusa_ipc_permission`
    - `perms=` množina vyžiadaných oprávnení
    - `ipc_class=` číselná reprezentácia typu IPC triedy
7. `medusa_ipc_semop`
    - `flag=` príznaky operácie
    - `sem_num=` index v poli semaforu
    - `sem_op=` semafor operácia
    - `nsops=` číslo operácie, ktorá sa má vykonať
    - `alter=` označuje, kde sa majú vykonať zmeny v semaforovom poli
    - `ipc_class=` číselná reprezentácia typu IPC triedy

8. `medusa_ipc_shmat`
    - `flag=` príznaky operácie
    - `shmaddr=` adresa, na ktorú sa má pripojiť pamäťový región
    - `ipc_class=` číselná reprezentácia typu IPC triedy
9. `medusa_link`
    - `rname=` vyžiadané meno
10. `medusa_mknod`
    - `mode=` file mód nového súboru
    - `dev=` číselná reprezentácia zariadenie, v hexadecimálnej sústave
11. `medusa_permission`
    - `mask=` obsahuje masku oprávnení
12. `medusa_readlink` – momentálne neaplikuje žiadne `post_audit` volanie
13. `medusa_rename`
    - `name=` aktuálne meno adresára alebo súboru
    - `rname=` vyžiadané meno
14. `medusa_rmdir`
    - `name=` aktuálne meno adresára alebo súboru
15. `medusa_symlink`
    - `path=` obsahuje meno cesty k súboru