UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

LUCRARE DE DIPLOMĂ

Coordonator științific:
ș.l.dr.ing. Cristian-Mihai AMARANDEI

Absolvent:
Fabian-Vlăduț BUZĂU

Iași, 2023

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Securitatea sistemelor Linux pe baza unor politici de control centralizate: Modulul de autorizare a aplicațiilor

Coordonator științific:
ș.l.dr.ing. Cristian-Mihai AMARANDEI

Absolvent:
Fabian-Vlăduț BUZĂU

Iași, 2023

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ

Subsemnatul BUZĂU FABIAN-VLĂDUȚ, legitimat cu C.I. seria NZ nr. 030101, CNP 5010101270831 , autorul lucrării: SECURITATEA SISTEMELOR LINUX PE BAZA UNOR POLITICI DE CONTROL CENTRALIZATE: MODULUL DE AUTORIZARE A APLICAȚIILOR elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii: CALCULATOARE organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi" din Iași, sesiunea din Iulie a anului universitar 2023, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi" din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data
06.07.2023

Semnătura

Cuprins

Securitatea sistemelor Linux pe baza unor politici de control centralizate: Modulul de autorizare a aplicațiilor

Fabian-Vlăduț BUZĂU

Rezumat

În aceasta lucrare, se aduce în discuție implementarea unui sistem de securitate bazat pe politici de control. Modulul dezvoltat are drept scop gestionarea și autorizarea aplicațiilor. Acesta este proiectat pentru a permite o administrare eficientă a aplicațiilor și pentru a garanta că doar aplicațiile autorizate pot fi utilizate în sistem. Deoarece se încadrează în domeniul securității cibernetice, obiectivul țintă este prevenirea unui atac la nivel de aplicație prin intermediul politicilor de acces.

Pentru implementarea și aplicarea politicilor de acces am utilizat demonul fapolicyd. Aceasta aplicație rulează ca un serviciu în fundal și monitorizează în timp real fiecare operație de acces la fișierele sistemului. Atunci când o cerere de acces la fișier este efectuată, fapolicyd analizează regulile definite în cadrul politicii și decide dacă operația este permisă sau nu. Aceste politici sunt definite de administratorul sistemului și pot fi modificate și personalizate în funcție de necesitățile acestuia.

De asemenea, pentru a proteja securitatea sistemului și în afara politicilor de control am implementat un modul de scanare care îi permite administratorului să vadă toate executabilele existente în sistem, ale căror proveniența este incerta și reprezinta un pericol pentru integritatea sistemului. Mai departe va rămâne la latitudinea administratorului dacă va restricționa sau nu accesul la executabilele respective.

Modul de funcționarea al lucrării va consta în interacționarea dintre doua module diferite: modulul de comanda și modulul de execuție care vor comunica intre ele prin intermediul unui serviciu D-bus. Aceasta comunicare va avea ca principal scop modificarea fișierelor de configurare a demonului Fapolicyd în funcție de ce presupune modulul de comanda.

Aplicația a fost dezvoltata în cadrul unui server de Centos 8 cu ajutorul limbajului de programare python. Momentan poate fi testata doar în interiorul distribuțiilor Red Hat, pe distribuția Debian neregăsindu-se încă o variantă stabilă a demonului Fapolicyd.

Prin urmare, scopul acestei lucrări este de a implementa un modul de securitate capabil sa ofere o gestionare eficienta a aplicațiilor în conformitate cu politica primita evitând astfel un posibil atac la nivel de aplicație.

Introducere

În ultimii ani întreaga industrie IT a lumii a pus accent pe dezvoltarea de produse software și hardware menite sa fie soluții pentru problemele utilizatorilor de rand, dar și pentru corporațiile de mici și mari dimensiuni ale globului. Pe de alta parte, odată cu evoluția dinamica a acestei industrii, au apărut și anumite grupări distructive menite sa atace datele personale ale utilizatorilor prin diferite tipuri de atacuri: Phishing, DDoS (Distributed Denial of Service), Malware etc.

Tema prezentei lucrări de licență, intitulată "Securitatea sistemelor Linux pe baza unor politici de control centralizate, modulul de autorizare al aplicațiilor", se încadrează în domeniul securității cibernetice. Are ca principal obiectiv combaterea unui atac cibernetic la nivel de aplicație, utilizând politici de acces. Exista un interes din ce în ce mai mare în ceea ce privește securitatea cibernetica și amenințările care apar odată cu aceasta, de aceea poate fi considerata un subiect de actualitate.

Atacurile de tip Malware consta în introducerea unui software râu intenționat în sistem în scop distructiv pe diferite cai precum: un fișier corupt, un e-mail de phishing, un device de tip USB infectat sau un site-web maliții os. Odată ajuns în sistem, obiectivul acestuia este de a accesa datele de confidențialitate ale persoanei, căpătând astfel drepturi depline asupra stației. De cele mai multe ori, atacatorii urmăresc sa capete acces la carduri bancare, efectuând tranzacții sau plați neautorizate și la diferite fișiere care pot fi vândute mai departe pe piața neagra. În majoritatea cazurilor, persoanele destinate acestui prejudiciu sunt obligate sa ofere o anumita suma de bani pentru recuperarea fișierelor furate.

Pentru evitarea unor posibile infiltrări în sistem a unui astfel de virus lucrarea propusa aduce în discuție conceptul de "securitate bazata pe politici de control" astfel încât controlul accesului la fișiere și executabile sa se facă pe baza acestor politici evitându-se lansarea în execuție a unor programe Râu intenționate care nu se regăsesc în politica care rulează pe stația respectiva. Totodată acest sistem de protecție va dispune și de o funcție de scanare a tuturor fișierelor executabile de pe întreg sistemul de operare verificându-se proveniența acestora. În cazul în care se identifică un executabil care are o proveniență incerta, administratorul poate restricționa accesul la respectiva resursa, aceasta neputând fi accesată de niciun utilizator.

Pentru realizarea restricționării accesului la fișiere și directoare am folosit demonul `fa policyd`. Acest *daemon* a fost dezvoltat în cadrul distribuției RedHat Enterprise Linux, pe variantele de Linux provenite din familia Debian neexistând încă o varianta stabila a acestuia. Principala lui funcționalitate consta în aplicarea unor "politici bazate pe etichete", acesta rulând ca un proces ascuns care monitorizează toate accesările asupra fișierelor și executabilelor pentru a verifica dacă accesul la resursele respective este sau nu permis.

Având un punct de plecare, lucrarea debutează cu un prim capitol explicativ " Fundamentarea teoretica și documentarea bibliografica", care va fi prezentat dintr-o perspectiva cauzala. Mai precis, conform literaturii de specialitate existente voi prezenta o analiza concisa și comparativa a progreselor recente similar realizate cu tema propusa.

Cel de-al doilea capitol, intitulat "Proiectarea aplicației", vizează analiza platformei software pe care va fi realizata aplicația, se stabilesc modulele generale ale aplicației și interacțiunile dintre ele. Mai mult decât atât, pentru o mai buna realizarea a structurii, se vor avea în vedere avantajele și dezavantajele metodei alese pentru a lua soluția cea mai buna de implementare, dar și limitele impuse după care metoda va activa.

Pentru a asocia conceptele generale, teoretice propuse în capitolul anterior, cu aspectul aplicativ, capitolul trei denumit "Implementarea aplicației" se concentrează pe partea practică a lucrării. In acest capitol se va pune accentul pe prezentarea modalității prin care au fost implementate funcțiile descrise anterior, fiecare funcționalitate fiind explicata la nivel de detaliu.

În ultimul capitol,capitolul 4: "Testarea aplicației și rezultatele obținute" se va descrie sistemul de testare prin care am aplicat diferite politici de acces,introdus reguli independente de restul sistemului și scanare a tuturor directoarelor din sistem în căutarea fișierelor suspecte. Fiecare funcționalitate descrisa anterior va avea o prezentare detaliată, cu capturi de ecran din timpul rulării acestora.

Finalul lucrării este format din concluziile proprii, atent formulate în urma cercetării realizate, dar și de bibliografia aferentă. Concluziile concepute au la bază atât capitolul contextual-descriptiv, cat și partea aplicativă.

Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

1.1. Domeniul și contextul abordării temei:

Domeniul aferent lucrării mele de licență este securitatea cibernetică. Securitatea cibernetică reprezintă unul dintre cele mai mari și importante domenii din aria IT și joacă un rol esențial în experiența fiecărui utilizator cu orice sistem, atât software cât și hardware. Aceasta se ocupă cu protejarea tuturor infrastructurilor IT, a rețelelor de diferite tipuri și a datelor confidențiale împotriva amenințărilor cibernetice[1]. Aceasta presupune implementarea unor seturi de măsuri luate în funcție de atacul cibernetic căruia i se adresează.

De exemplu :

- Pentru asigurarea unui mediu online sigur și care nu este predispus unui atac la nivel rețea se recomandă utilizarea unui firewall care filtrează tot traficul din rețeaua respectivă în funcție de regulile definite.
- Pentru prevenirea unui atac cibernetic prin lansarea în execuție a unui executabil suspect în scop distructiv (atac de tipul Malware[2]), se recomandă utilizarea unui anti-virus care să ruleze în timp-real capabil să detecteze introducerea unui software malițios printre fișierele sistemului și să se ocupe cu eliminarea sau restricționarea accesului asupra acestuia.Aici se regăsesc mai multe opțiuni de anti-viruși printre care se numără și: ClamAV, Avast, Bitdefender GravityZone Business Security. O altă metodă de securitate care să combată acest tip de atac îl reprezintă utilizarea unei politici de control prin care, la fel ca în cazul firewall, un utilizator va putea accesa doar ceea ce îi este definit în politică. Un exemplu de software care ar putea fi configurat să implementeze astfel de politici este: `Fapolicyd`[3].
- O altă soluție pentru păstrarea confidențialității datelor unui sistem este Criptografia. Aceasta se ocupă cu criptarea datelor transformându-le într-o formă indescifrabilă de atacatori. Criptografia este necesară în transferurile de date importante, dar și în stocarea acestora.
- Printre cele mai răspândite atacuri cibernetice asupra utilizatorilor simpli, dar și către companiile mari din diferite domenii sunt atacurile de tip phishing[4].. Aceste atacuri constă în înșelătoria victimei printr-un mail (de obicei) în care atacatorul susține că are o identitate de încredere: fie o instituție bancară, un furnizor de servicii sau un direct partener de afaceri și cere de la victimă date confidențiale cum ar fi parole, date bancare în scopul utilizării acestora în scopuri ilegale.

  Metoda preferată de atacatori pentru phishing este introducerea într-un mesaj a unui link către o pagină web malițioasă unde, atunci când victima va accesa acea pagina, automat numele și parola victimei vor fi salvate. O altă posibilitate este ca atunci când accesezi siteul respectiv, să se instaleze un anumit software rău intenționat în zone ascunse din unitatea ta de lucru provocând astfel un atac de tipul Malware căpătând astfel drepturi totale asupra unității.

  Pentru combaterea acestor tipuri de infracțiuni se recomandă:

  - Instalarea unui software care se ocupă cu detectarea site-urilor de phishing și blocarea acestora;
  - Se recomandă utilizarea filtrelor spam pentru reducerea șanselor de a primi acest tip de mesaj;
  - Dacă mail-ul respectiv conține informații în care se cer date confidențiale, verifică autenticitatea acelui mail înainte de trimiterea acestor date;

- Utilizarea autentificării prin mai multe moduri. Acest lucru oferă o securitate mai bună asupra conturilor în sine, făcându-le mult mai greu de accesat, dacă nu chiar imposibil în cazul unei breșe de securitate;
- Nu in ultimul rând, în cazul în care ai identificat un anumit mesaj de tipul phishing este esențial să-l raportezi furnizorului de servicii.

1.2. Tema propusă

Lucrarea prezentată în cadrul acestei licențe încearcă să vină ca o soluție pentru combaterea unui atac la nivel de aplicație prin implementarea unui modul de autorizare configurat, astfel încât să poată interpreta diferite politici de control asupra directoarelor și executabililor din sistem cu ajutorul "permisiunilor bazate pe etichete".

1.3. Subiecte și teme similare

Pe tema securității bazate pe politici de control au fost implementate mai multe aplicații regăsite pe sistemele de operare Linux, printre care se numără și:

- SELinux:

  A fost dezvoltat în cadrul sistemului de operare Linux de către United States National Security Agency (NSA), scopul lui fiind de a aduce un plus de securitate sistemelor de operare cu ajutorul politicilor de control. Acest software a fost implementat pentru a ajuta administratorul să dețină mult mai ușor controlul asupra sistemului pe care îl administrează.

  Modul de funcționare:

  - Principiul de funcționare al aplicației SELinux constă în atribuirea unor etichete asupra tuturor fișierelor,proceselor și resurselor existente pe sistemul de operare;
  - Dacă un utilizator care nu este administrator, sau un proces, încearcă să utilizeze o resursă din sistem, automat se va face o cerere unde se va verifica cu ajutorul unui cache vector de acces(AVC) dacă resursa definită este sigură sau nu;
  - Dacă SELlinux nu are definit nicio politică asupra acelei resurse definită în cache, va trimite cererea la serverul de securitate, acesta luând o decizie de tip *allow* sau *denied* asupra resursei;
  - În cazul în care accesul a fost refuzat în fișierul. `/var/log/messages` se va regăsi mesajul `"avc:denied"`

  Puncte tari:

  - SELinux oferă un nivel înalt de securitate cu ajutorul politicilor stricte bazate pe etichete;
  - Rulează în timp real și interoghează fiecare acces la resurse;
  - Detectează și previne atacurile de tip Malware;
  - Poate fi regăsit și pe sistemul de operare Android.

  Puncte slabe:

  - Este o aplicație destul de complexă și necesită un nivel de documentare destul de ridicat pentru a putea lucra cu acesta;
  - Din cauza volumului de date mare pe care le interoghează, acesta poate scădea nivelul de performanță al sistemului.

Se poate observa în Figura de mai jos funcționalitatea acestuia:

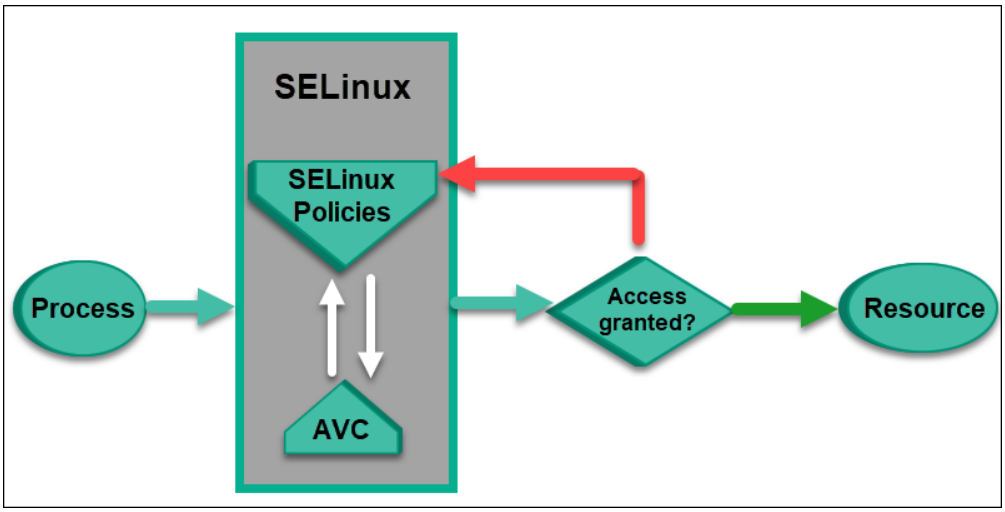


Figura 1.1. How-selinux-works[5]

- APParmor:

  Această aplicație a fost dezvoltată în cadrul sistemelor Linux și constă în aplicarea unui modul de securitate care utilizează controale de acces definite pe baza unor profiluri, fiecare profil dispunând de setul lui de reguli. Acest software este regăsit în general pe variantele de Linux provenite din familia Debian.

  Modul de funcționare:

  – Se declară profilurile necesare asupra fiecărei aplicații asupra căruia se dorește a se interveni;
  – Se declară regulile necesare fiecărui profil;
  – Pentru a învăța modul de utilizare al aplicației, dar și pentru definirea de noi profiluri a fost implementat un model de utilizare intitulat astfel: `Complaining / Learning`, acesta permite încălcări ale profilului și le înregistrează în jurnale.
  – Pentru aplicarea riguroasă a regulilor definite în profiluri se utilizează modelul `Enforced / Confined`, acest mod nu permite nicio încălcare a regulilor definite și înregistrează toate încălcările de acces.

  Puncte tari:

  – Din punct de vedere al securității la nivel de aplicație, acesta oferă un nivel înalt de securitate dacă regulile de securitate sunt definite corespunzător;
  – Pe lângă regulile definite asupra accesului la fișierele sistemului, se pot defini reguli și pentru a restricționa anumiți utilizatori care folosesc aplicația respectivă.

  Puncte slabe:

  – Datorită definirii unui profil pentru fiecare aplicație, necesită un volum de muncă destul de ridicat dacă îți dorești să restricționezi mai multe aplicații;
  – Dacă profilurile declarate nu sunt bine definite sau incomplete, pot exista anumite vulnerabilități care pot fi utilizate de atacatori.

Se poate observa în schema de mai jos funcționalitatea acestuia:

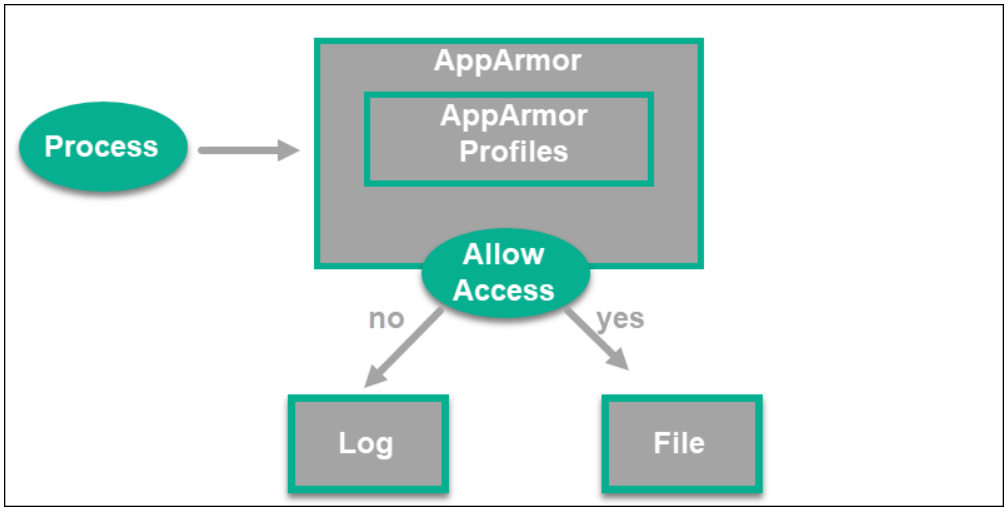


Figura 1.2. apparmor-work-flow[6]

- SMACK (Simplified Mandatory Acces Control Kernel):

  Acest software reprezintă un modul de securitate implementat în kernelul Linux, care se bazează pe controlul accesului la fișierele sistemului prin intermediul etichetelor. Se bazează în mare parte pe aceleași funcționalități ca software-ul SELinux prezentat anterior.[7]

  Modul de funcționare:

  – Modulul prezentat se bazează pe utilizarea etichetelor, astfel pentru restricționarea unei resurse, trebuie declarată o regulă prin care această resursă este marcată ca fiind *trust / untrust*.

  – După definirea regulilor, fiecare încercare de acces la o resursă, fie fișier,executabil sau proces, va fi urmărită și verificată conform regulilor definite. În cazul în care resursa este marcată ca fiind sigură se permite accesul, iar dacă aceasta e marcată ca fiind nesigură utilizatorul nu o va putea accesa.

  Puncte tari:

  – Oferă un mod de lucru și configurare prietenos administratorului, fiind un software care nu necesită cunoștințe avansate;

  – Se regăsește pe toate distribuțiile de Linux;

  – Poate fi folosit în implementarea unor politici complexe de securitate.

  Puncte slabe:

  – Pentru o securitate avansată fără vulnerabilități este necesară utilizarea unei politici bine definite;

  – Existența unor diferite aplicații mai mult utilizate atât pe variantele de Linux din familia RedHat, unde predomină SELinux, cât și pe variantele de Linux provenite din Debian, unde predomină APParmor.

1.4. Resurse Software utilizate în lucrare

1.4.1. Fapolicyd

Principalul software pe care l-am utilizat pentru autorizarea aplicațiilor este demonul **Fapolicyd**, deoarece din punct de vedere al configurației și al modului de lucru, oferă mult mai multă flexibilitate, astfel încât poate fi utilizat pentru implementarea politicilor de control.

Fapolicyd a fost dezvoltat în cadrul sistemelor de operare Linux și are ca principal scop securitatea sistemului de operare bazată pe utilizarea politicilor de control. Acest software poate fi configurat să implementeze diverse politici, inclusiv politici de tipul "permisiuni bazate pe etichete" pentru a controla accesul la fișiere în funcție de anumite reguli definite de administratorul de sistem. Implementarea acestui software a avut loc pe distribuția de linux RedHat, regăsindu-se într-o variantă stabilă doar pe aceste variante de Linux, pe distribuția Debian neexistând încă o implementare stabilă a acestuia.

Pentru instalarea acestui software am folosit comanda:

```
# yum install fapolicyd
```

Structura acestuia se regăsește în folderul /usr/etc/fapolicyd și este formată din :

- `compiled.rules`: acest fișier conține toate regulile de acces definite și care rulează pe stație.
- `fapolicyd.conf`: Acest fișier controlează configurarea politicii de acces.
- `fapolicyd.trust`: Aici sunt preluate și stocate toate fișierele găsite în folderul trust.d.
- `trust.d`: În acest director se vor stoca path-urile către fișierele trust.
- `rules.d`: Acest director conține o listă de fișiere unde se regăsesc toate regulile predefinite cu denumiri sugestive asupra cărora se poate interveni atunci când dorim sa adăugam sau să scoatem anumite reguli: `10-languages.rules, 20-dracut.rules, 21-updaters.rules, 30-patterns.rules, 40-bad-elf.rules, 41-shared-obj.rules, 42-trusted-elf.rules, 70-trusted-lang.rules, 71-shell.rules, 90-deny-execute.rules, 95-allow-open.rules`

  Acest director este pricipalul spațiu de lucru cu care vom lucra în definirea regulilor, fiecare fișier cu extensia "`.rules`" putând fi compilat de aplicație.

Pentru a putea utiliza fapolicyd este necesară cunoașterea următorului set de comenzi:

- Pentru pornirea aplicației am folosit comanda:

  `# systemctl enable -now fapolicyd` sau `#systemctl start fapolicyd`

- Pentru oprirea aplicației am folosit comanda:

  `# systemctl stop fapolicyd`

- Pentru a verifica dacă în folderul `rules.d` au avut loc schimbări:

  `# fagenrules -check`

- În cazul în care au avut loc schimbări, pentru a adăuga noile schimbări pot fi folosite următoarele comenzi:

  `# fagenrules -load` sau `#fapolicyd-cli -update`

- În cazul în care întâmpinăm erori la compilarea regulilor, trebuie utilizată următoarea comandă:

  `# fapolicyd -debug`

  Această comandă va identifica unde a apărut eroarea și va informa utilizatorul în legătură cu regula definită incorect.

- Dacă dorim să vizualizăm toate regulele compilate care rulează pe stație utilizăm următoarea comandă

  ```
  # fapolicyd-cli -list
  ```

Structura unui reguli Fapolicyd este : `decision perm subject : object`, fiecare termen din regulă prezentată putând fi configurată astfel: [8]

- `Decision`:

  Acesta reprezintă decizia luată asupra regulii definite și poate avea următoarele atribute `allow, deny, deny_audit, allow_audit, allow_log, deny_log, allow_syslog, deny_syslog`

- `Perm`: Acest atribut reprezintă permisiunea care va fi utilizată în cadrul regulii și poate fi de 3 tipuri:

  - `Open`: Este folosit în general pentru directoare sau pentru diferite fișiere;
  - `Execute`: Este folosit asupra executabilelor și în funcție de decizie, permite sau nu lansarea în execuție a programului;
  - `Any`: Este folosit când se dorește o restricționare completă asupra unui director, fișier sau executabil.

- `Subject`:

  În acest atribut se regăsesc constrângerile impuse de regulă. Aici se specifică user-ul sau grupul căruia dorim să-i atribui o constrângere. Subiectul poate avea o gamă variată de caracteristici precum: `all` (aici se regăsesc toate combinațiile de constrângeri existente), `auid, uid (user), gid (group), sessionid, pid, ppid , trust, comm, exe, dir, ftype, device, pattern`.

- `Object`: Aici se regăsește resursa asupra căruia se aplică constrângerile definite anterior. Obiectele pot fi de mai multe tipuri:

  - `all`: toate obiectele existente în sistem;
  - `path`: locația unui singur fișier;
  - `device`: aici se definește un obiect de tipul device astfel: `/dev/` și numele dispozitivului;
  - `ftype`: toate obiectele care sunt de tipul definit.

Fapolicyd atunci când este instalat pentru prima dată vine la pachet cu un set de reguli care afectează sistemul în modul următor:

- `allow perm=any uit=0:dir=/var/tmp` - Această regulă permite doar administratorului să acceseze directorul `/var/tmp`
- `allow perm=any uid=0 trust=1 :all` - Permite administratorului să acceseze orice resursă
- `allow perm=open exe=/usr/bin/rpm : all` - Permite accesul la executabilul rpm pentru toate obiectele
- `allow perm=open exe=usr/1ibexec/platforn-python3.6 comm=dnf: all` - Permite accesarea executabilului python atunci când este chemat de comanda `dnf`

- `deny_audit perm=any pattern=ld_so : all` - Refuză orice acțiune ce presupune accesarea fișierelor care conțin `ld_so` în calea lor
- `deny_audit perm=any all : ftype=application/x-bad-elf` - Refuză orice acțiune cu tipul "`application/x-bad-elf`"
- `allow perm=open all : ftype=application/x-sharedlib trust=1` - Această regulă marchează toate fișierele de tipul `application/x-sharedlib` ca fiind de încredere
- `deny_audit perm=open all : ftype=application/x-sharedlib` - Această regulă marchează toate fișierele de tipul `application/x-sharedlib` ca fiind de neîncredere
- `allow perm=execute all : trust=1` - Această regulă permite execuția tuturor fișierelor
- `allow perm=open all : ftype=%languages trust=1` - Permite accesul la orice fișier de tipul " `%languages` "
- `deny_audit perm=any all : ftype=%languages` - Această regulă refuză orice acțiune cu fișierele de tip "`xlanguages`"
- `allow perm=any all : ftype=text/x-shellscript` - Permite accesul la toate fișierele de tipul shell script
- `deny_audit perm=execute all : all` - Nu permite utilizatorilor non-root să ruleze niciun fel de executabil
- `allow perm=open all : all` - Permite citirea oricărui fișier de catre toti utilizatorii

Pentru o bună funcționare a sistemului, am eliminat regulile prezentate mai sus deoarece afecteaza functionalitatea normala a sistemului de operare.

Arhitectura Fapolicyd este următoarea:

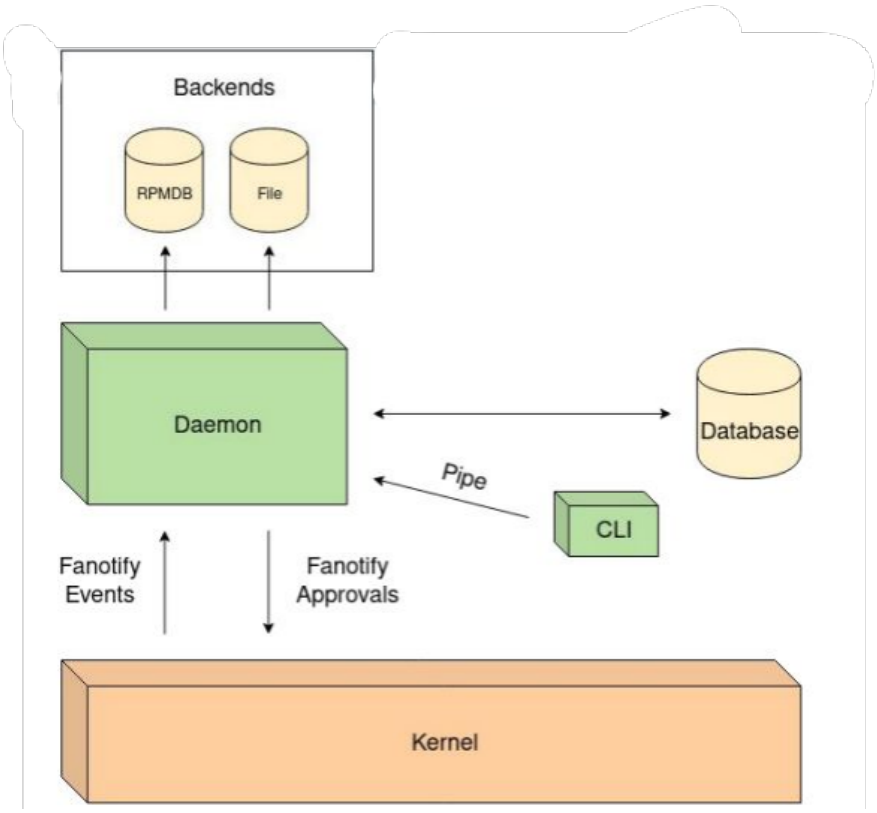


Figura 1.3. Fapolicyd-arhitecture[9]

1.4.2. Python

Python este unul dintre cele răspândite limbaje de programare existente la timpul actual și se remarcă de alte limbaje de programare prin simplitatea și sintaxa ușor de învățat. Acest limbaj a fost dezvoltat în anul 1991 de către Guido vas Rossum, extinzându-se de atunci până în prezent pe toate sistemele de operare. Datorită simplității și eficienței acestuia, majoritatea companiilor din ziua de azi aleg să-l folosească în diferite domenii:

- Dezvoltare de software;
- Administrarea și automatizarea sistemelor;
- Web development;
- Dezvoltare de jocuri etc.

În lucrarea mea de licență, am ales să utilizez versiunea de python 3.6.8 în scopul automatizării întregului proces de aplicare al politicilor, dar și pentru implementarea de diverse funcționalități.

Printre punctele tari ale acestui limbaj pe care le-am sesizat pe parcursul realizării lucrării se numără: existența a nenumărate biblioteci destinate automatizării sistemelor precum: `os, json, grp, dbus`, toate acestea ajutând în procesul de prelucrare a datelor.

1.4.3. Oracle VM VirtualBox

Oracle VM VirtualBox este un software dezvoltat de către Oracle Corporation care ajută la crearea mașinilor virtuale. Aceste mașini virtuale pot găzdui diferite sisteme de operare de la Windows, Linux, macOS până la Oracle Solaris și Android. În cadrul implementării acestei lucrări Oracle VM VirtualBox a jucat un rol foarte important, deoarece cu ajutorul acestuia am simulat un server cu Centos8, pe care de altfel, am și realizat lucrarea. Setările mașinii virtuale utilizate au fost:

- Base Memory (RAM) = 1024
- 1 CPU
- Video memory = 16

1.4.4. Dbus

Desktop Bus reprezintă un sistem de comunicare între procesele unui sistem de operare și a fost dezvoltat pentru a aduce o îmbunătățire comunicării dintre diferite aplicații în cadrul sistemelor Linux. Comunicațiile în cadrul D-bus sunt de tipul "mesaj" , D-bus fiind mai exact o magistrală pe care circulă mesaje provenite de la diferite aplicatii-uri. Acest tip de comunicație ajută în primul rând la lansarea în execuție a aplicațiilor și a demonilor atunci când se cere activarea acestora D-bus se remarcă prin faptul că dispune de biblioteci pentru o gamă variată de limbaje de programare: Python, Java, C, C++ etc. Astfel pentru configurarea acestui D-bus, am ales să folosesc modulul dbus din Python.

Modul de funtionare al comunicatiilor pe canalul Dbus:

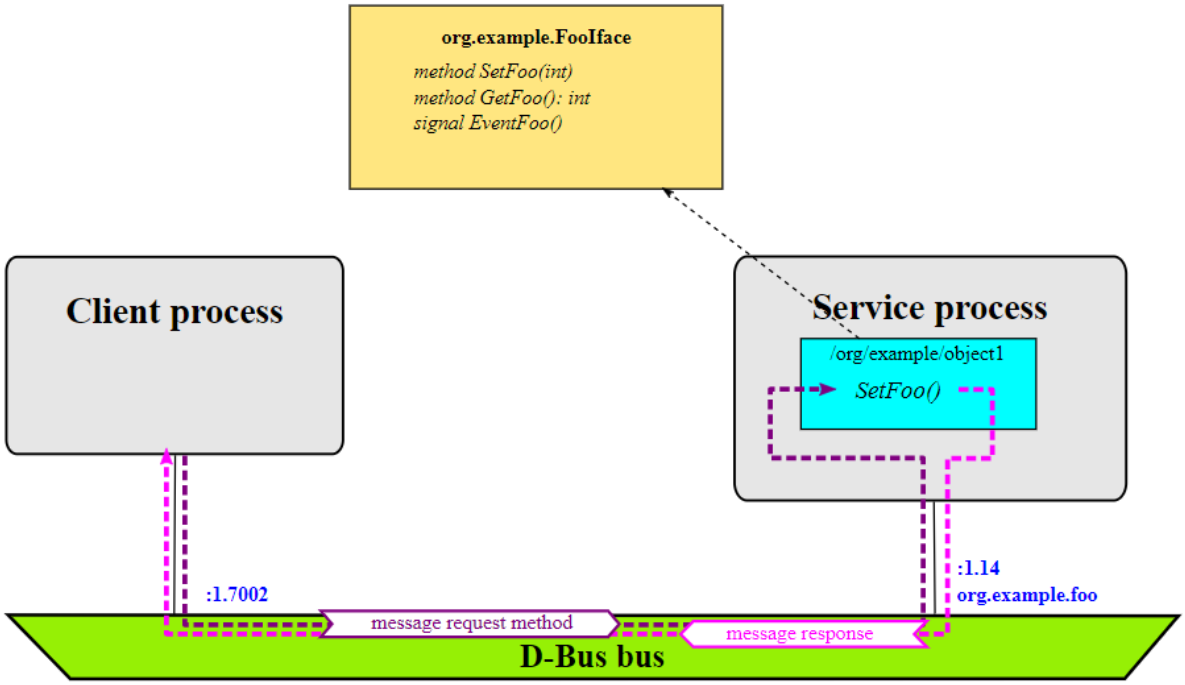


Figura 1.4. Dbus_comunication[10]

După cum se poate observa în figura de mai sus, clientul face un request asupra metodei SetFoo regăsite în canalul de comunicații la adresa, obiectului `/org.example.object1`. Obiectul apelează metoda definita și trimite înapoi pe canalul dbus un răspuns cu privire la metoda apelată. D-bus poate fi de două tipuri:

- D-bus de sistem: Acest canal permite comunicarea între serviciile de sistem și aplicațiile utilizatorului. În general, acesta tratează evenimente legate de partea hardware (introducerea unui dispozitiv noi).
- D-bus de sesiune: Acesta constă într-un canal D-Bus privat asignat unui singur utilizator și se ocupă cu comunicațiile dintre procesele aplicațiilor.

În această lucrare D-bus a jucat un rol foarte important, deoarece a ajutat la realizarea comunicațiilor dintre module. Prin intermediul acestuia, am reușit să configurez setările demonului Fapolicyd, dar și să execut diferite comenzi de pe sistemul de operare.

1.5. Obiective

Acest modul de autorizare va trebui să respecte următoarele funcționalități:

- Implementarea politicilor de control cu ajutorul fapolicyd
- Implementarea unei opțiuni de detectare de executabile din surse necunoscute
- Implementarea unui modul de testare cu care se poate manipula starea sistemului în timp real. Acest modul va trebui să aibă următoarele funcționalități
  - Pornirea și Oprirea demonului fapolicyd;
  - Aplicarea unei politici la alegerea administratorului;
  - Inserarea unei reguli separate de restul sistemului;
  - Scanarea tuturor executabililor existenți în cadrul unui director dat;
  - Funcționalitatea de reset care aduce fapolicyd la starea lui inițial configurată.

algorithmic

Capitolul 2. Proiectarea aplicației

2.1. Arhitectura aplicației

Aplicația prezentată în această lucrare a fost realizată conform arhitecturii prezentate mai jos:

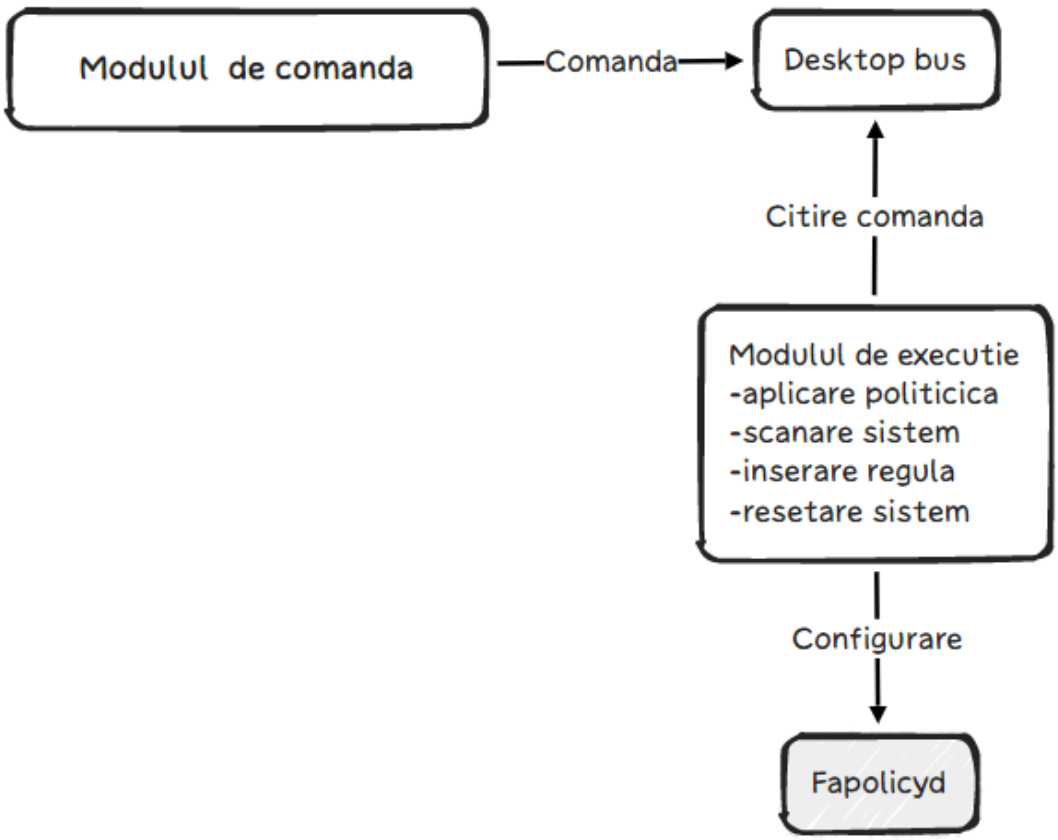


Figura 2.1. Arhitectura aplicației

După cum reiese din arhitectura prezentată mai sus, structura lucrării va fi bazată pe doua module separate care comunică între ele cu ajutorul Dbus. Comunicarea dintre aceste două module va avea ca principal obiectiv configurarea demonului fapolicyd, care va fi supus la diferite modificări în funcție de ce cerință presupune modulul de comandă.

Modul în care funcționează sistemul:

- Se pornește serverul de Dbus in cadrul unei distribuții de Linux provenite din familia Red Hat și se concretizează un obiect de tipul MyDbusObject
- Se pornește modulul de execuție care monitorizează în timp real ce semnale circulă pe dbus și în funcție de ce semnale primește, modifică starea sistemului.
- Se trimit comenzi pe Dbus prin intermediul modulului de comandă, apelând metodele definite activării semnalelor.
- Semnalele fiind active, va interveni modulul de execuție ,care preia datele comenzii și modifică starea demonului fapolicyd în funcție de comanda primită.

Pentru a putea utiliza această aplicație pe orice stație de lucru, este nevoie de drepturi de administrator pe stația respectivă, scopul acesteia fiind să-i atribuie administratorului controlul absolut și să gestioneze mai ușor accesul la fișierelede pe mașină.

2.2. Diagrama UML și diagrama de Secvențe

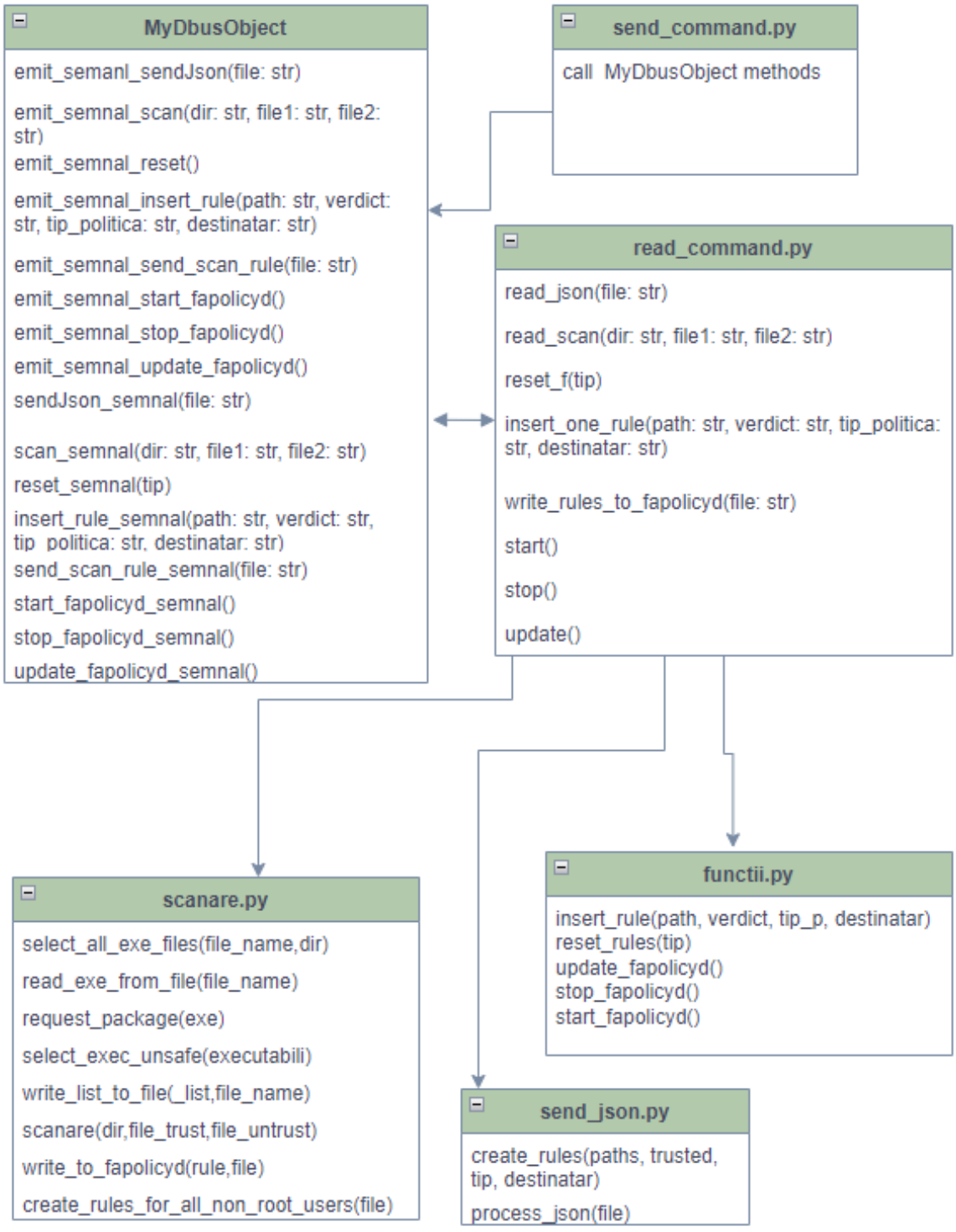


Figura 2.2. Diagrama UML

Diagrama UML prezentată mai sus este formată din metodele clasei MyDbusObject și toate funcțiile din fiecare modul implementat. Din structura prezentată reiese și conectivitatea modulelor între ele și modul în care acestea se apelează.

Modulul de comandă este reprezentat de programul `send_command.py` care apelează o metodă din clasa `MyDbusObject`. Aceasta la rândul ei, emite un semnal conform metodei apelate, semnal care va fi citit și recunoscut de modulul de execuție care este reprezentat de programul `read_command.py`. Ulterior, modulul de execuție va apela toate functionalitățile existente în programele `scanare.py, functii.py, send_json.py` care vor avea un impact direct asupra demonului Fapolicyd. Modul în care acestea sunt conectate, reiese și din diagrama de secvențe prezentată mai jos:

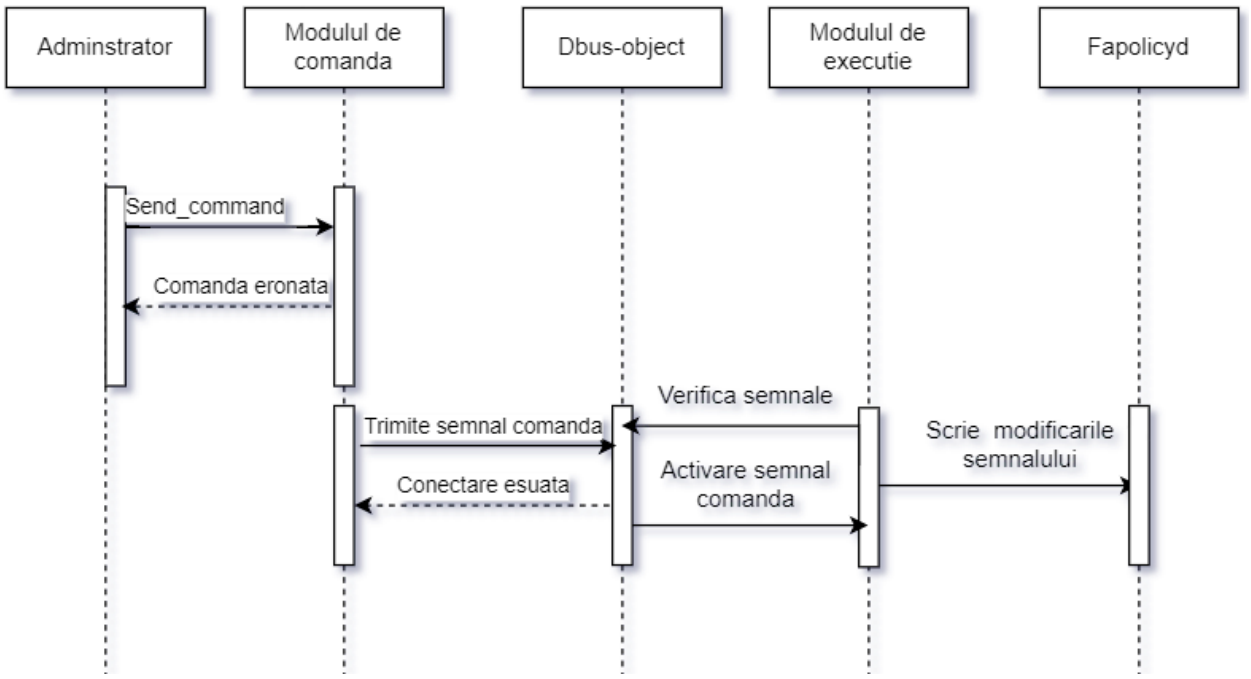


Figura 2.3. Diagrama de secvențe

2.3. Componente

2.3.1. Modulul de comandă

Modulul de comandă a fost realizat cu ajutorul limbajului Python și asigură transmiterea de mesaje de la administrator, la server-ul de Dbus. Caracteristici generale:

- Reprezintă un executabil Python care este executat cu ajutorul liniei de comandă.
- Este configurat să ruleze în două moduri:
  - Un mod autonom în care nu este nevoie de intervenția administratorului prin intermediul liniei de comandă și al argumentelor trimise;
  - Un mod care nu este capabil să ruleze autonom și este nevoie de intervenția administratorului, deoarece este nevoie de o putere de decizie. Acest mod este utilizat doar atunci când se apelează o singură comandă care necesită niște răspunsuri din partea administratorului cu privire la securitatea sistemului.
- Monitorizează datele trimise prin intermediul argumentelor și ia decizii în funcție de valorile acestora

- În cazul introducerii unei date eronate, administratorul va fi informat printr-un mesaj în care i se definesc toate comenzile disponibile și toate argumentele necesare rulării acestora;
- Comunicarea cu serverul de Dbus a fost realizată prin intermediul unui obiect de tipul: `dbus.service.Object`;
- Pentru a se conecta la Dbus, programul citește dintr-un fișier config_file.txt următoarele: identificatorul serviciului DBus, adresa obiectului DBus și identificatorul interfeței DBus;
- Trimiterea de semnale pe Dbus a fost îndeplinită cu ajutorul metodelor unei interfețe direct conectate la obiectul de tip dbus.service.Object.

Comenzi disponibile:

- Comanda pentru trimiterea unei politici la obiectul Dbus. Această comandă poate fi apelată direct din linia de comandă și trebuie să primească ca parametrii, identificatorul de regulă:"send_politica"un și un path către o politică din directorul de politici existent în lucrare.

  Exemplu:

  ```
  $ ./send_command.py send_policy name_of_policy
  ```

  sau

  ```
  $ ./send_command.py scan_system /path_to_dir /path_all_exe
  /path_untrust_exe
  ```

- Comanda pentru scanarea sistemului pentru identificarea executabililor care reprezintă un posibil risc pentru sistem. Această comandă primește ca argumente 4 parametrii: identificatorul de comandă "scan_system", path-ul directorului asupra căruia i se va aplica scanarea, path-ul unui fișier txt destinat scrierii adreselor tuturor executabililor și path-ul altui fișier unde vor fi stocate adresele executabililor suspecti. În cazul în care se va specifica doar directorul, comanda va scrie executabilele în două fișiere prestabilite existente în folderul scan_files din cadrul lucrării.

  Exemplu:

  ```
  $ ./send_command.py scan_system /path_to_dir
  ```

  sau

  ```
  $ ./send_command.py scan_system /path_to_dir /path_all_exe
  /path_untrust_exe
  ```

- Comanda pentru inserarea unei reguli pentru demonul fapolicyd. Acest tip de comandă primește 5 argumente: un identificator de comandă:"insert_rule", un path către un director sau executabil, un verdict de sigur sau nesigur(1 sau 0),tipul de destinatar asupra căruia i se adresează user/group și numele user-ului sau grupului de useri.

  Exemplu:

  ```
  $ ./send_command.py insert_rule /path_dir/exe 0 user student1
  ```

  sau

  ```
  $ ./send_command.py insert_rule /path_dir/exe 0 group studenti
  ```

- Comanda pentru crearea și scrierea regulilor provenite în urma unei scanări. Această comandă necesită două argumente: un identificator de comandă: "create_scan_rules" și un path către un fișier txt. În cazul în care lipsește path-ul către fișierul txt, se va folosi unul predefinit.

Exemplu:

```
$ ./send_command.py create_scan_rules /path_to_file_txt
```

- Comenzi pentru manipularea directă a demonului Fapolicyd. Acestea vin doar cu un singur argument și acela fiind indicatorul de comandă:"`start/stop/update_fapolicyd`"

  Exemplu:

  ```
  $ ./send_command.py start/stop/update_fapolicyd
  ```

2.3.2. Modulul de executie

Modulul de execuție se va ocupa cu interogarea constantă a Dbus-ului, în scopul recepționării de semnale, iar toate funcționalitățile sistemului vor fi apelate prin intermediul acestui modul. În această parte se definesc niște funcții capabile să monitorizeze diferite semnale definite pe o interfață Dbus prin intermediul funcției: `connect_to_signal(function,signal)` apelată de către o instanță a unui obiect de tipul `dbus.Interface`.

Fiind un modul de execuție, acesta va fi cel care va aduce modificări sistemului aplicând funcțiile activate și va reprezenta puntea de legătură dintre aplicație și Fapolicyd.

2.3.3. Componenta Dbus

După cum se poate observa în arhitectura sistemului, Dbus-ul joacă un rol esențial în funcționalitatea sistemului. În cadrul acestui modul se va instanția un obiect de tipul `MyDbusObject` asociat unor configurații predefinite pentru recunoașterea acestui modul de toate componentele. Acest obiect va dispune de o serie de metode și semnale care să facilizeze traficul de date de pe canalul de comunicații. Fiecare metodă va fi conectată la un semnal pe care îl va activa atunci când aceasta va fi apelată.

Modul în care are loc comunicația dintre administrator și Dbus este prezentat în Figura 2.4:

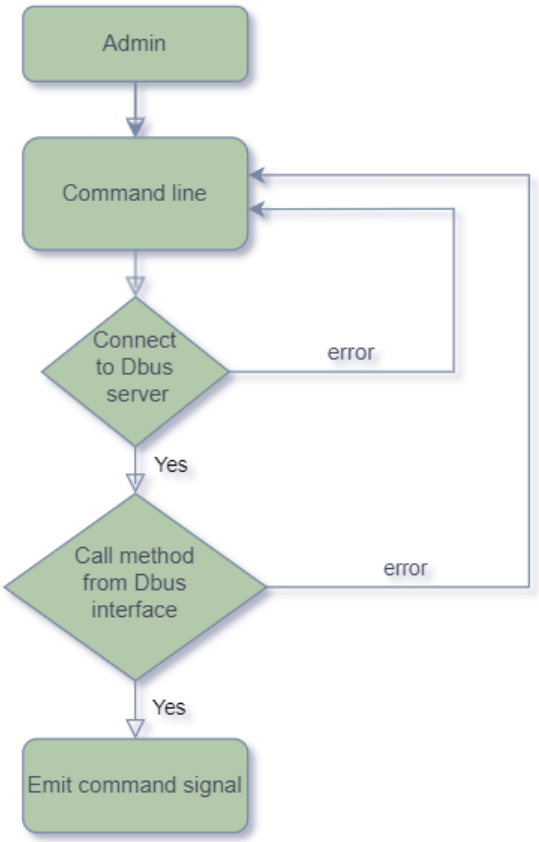


Figura 2.4. Cmunicație Administrator-Dbus

2.3.4. Componenta Fapolicyd

Fapolicyd este obiectul principal din cadrul acestei lucrări, deoarece el este destinația tuturor funcționalităților implementate. Interacțiunea aplicației cu acest demon se va realiza prin implementarea de reguli, în diferite contexte, care vor fi scrise în spațiul de configurare aflat la adresa "`/etc/fapolicyd/rules.d`". Aici se vor regăsi doar fișiere cu extensia "`.rules`" , acestea conținând regulile care se aplică în timp real pe sistem. Pentru activarea acestuia am folosit sistemul de gestionare a serviciilor și proceselor "systemd"[11] utilizând comanda "`systemctl`".

2.4. Algoritmi

2.4.1. Algoritm Scanare

Algoritmul funcției de scanare va fi implementat conform pseudocodului următor:

```
function SELECT_ALL_EXE_FILES(file_name, dir)
    command ← "find dir -type f -executable"
    ruleaza comanda "command » file_name"
function REQUEST_PACKAGE(exe)
    command ←"rpm -qf exe"
    result ← executa_comanda(command)
    if "not owned by any package" in result then
        return True
    else
        return False
function SELECT_EXEC_UNSAFE(executabili)
    exec_untrust← lista_vida()
    for executabil in executabili do
        if request_package(executabil) then
            exec_untrust.append(executabil)
    return exec_untrust
function SCANARE(dir, file_exe, file_untrust)
    golește conținutul fișierelor file_exe si file_untrust
    select_all_exe_files(file_exe, dir)
    executabili ←citeste_exe_din_fisier(file_exe)
    exec_untrust ← select_exec_unsafe(executabili)
    scrie_lista_in_fisier(exec_untrust, file_untrust)
```

Etapele scanării care reies din algoritmul prezentat mai sus sunt:

- Curățarea fișierelor în care urmează să se scrie adresele executabililor
- Funcția `select_all_exe_file(file_name,dir)` folosește comanda `find dir -type f -executable » file_name`[12]. pentru a selecta toți executabilii din sistem, din directorul dir și din subdirectoarele acestuia și a le scrie în fișierul stocat la adresa file_name.
- În lista executabili, se vor stoca adresele scrise anterior cu ajutorul unei funcții de citire: `citeste_exe_din_fisier(file_exe)`
- Lista de executabili va fi trimisă mai departe pentru verificare prin intermediul funcției `select_exec_unsafe(executabili)`, unde pentru fiecare executabil citit se va apela o funcție denumită `request_package(exe)`, funcție care va returna True sau False în funcție de proveniență executabilului. Dacă executabilul are o proveniență sigură, mai exact, dacă la execuția comenzii "`rpm -qf exe`" [13] obținem un repository cunoscut sistemului va returna True, iar în caz contrar False. Fiecare executabil suspect va fi stocat într-o altă listă pe care funcția o va returna
- Lista `exec_untrust` va reprezenta lista returnată în urma scanării, pe care o vom scrie într-un fișier txt cu ajutorul funcției `scrie_lista_in_fisier(lista,file_name)`

Separat de partea de scanare, unde doar am selectat executabilele care nu-și pot justifica proveniența, am implementat un modul separat care se ocupă cu prelucrarea acestor date printr-o interacțiune directă cu administratorul conform arhitecturii prezentate în următoarea figură:

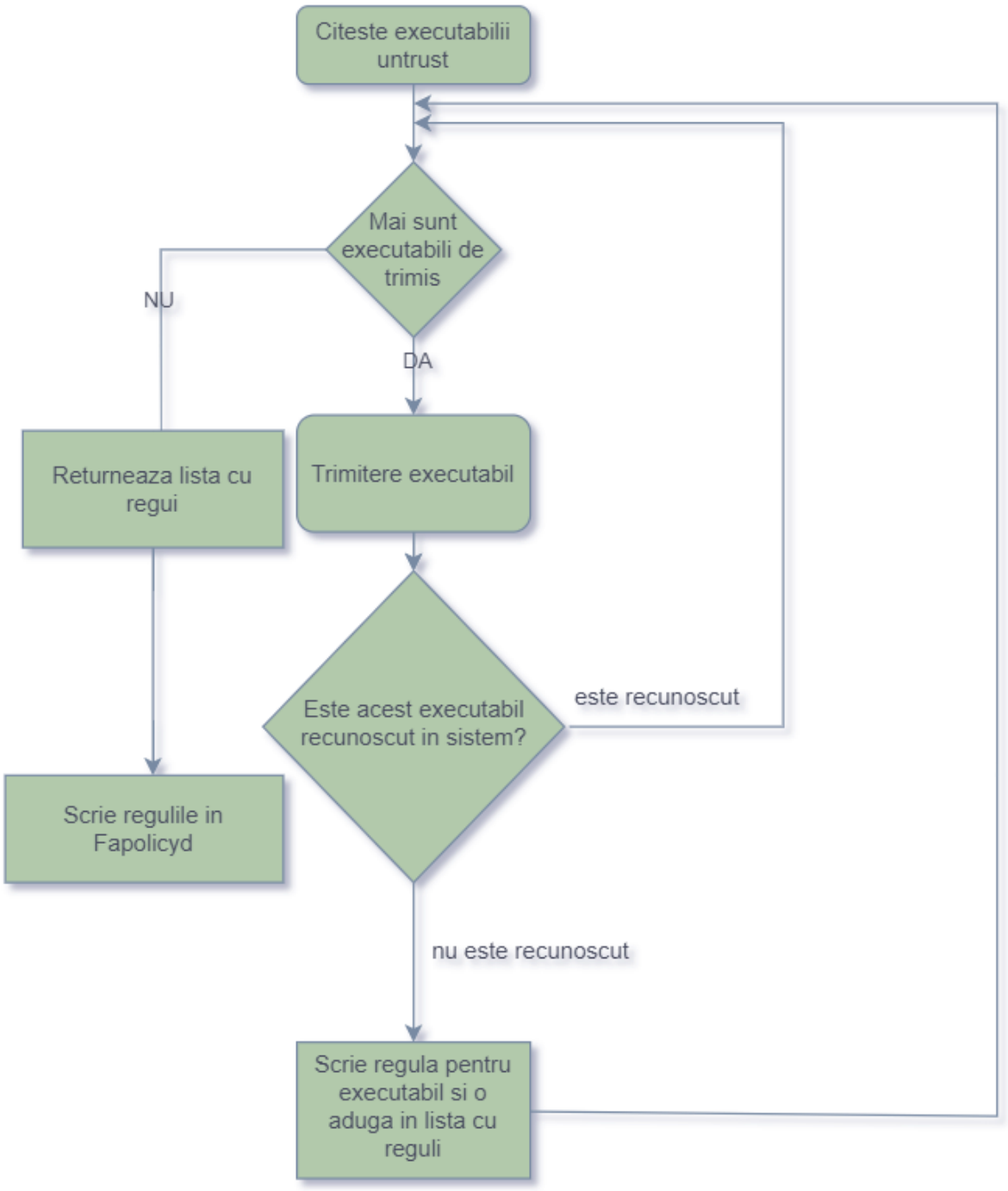


Figura 2.5. Workflow_untrust_scan_exe

Deoarece proveniența executabilelor poate fi recunoscută de administrator, nu s-a putut face o marcare a tuturor acestor executabili găsiți, ca fiind suspecți în mediul fapolicyd, aceasta nefiind o metoda eficientă putând să afecteze într-un mod sau altul integritatea sistemului. Astfel, am ales să implementez o comunicație directă cu administratorul sistemului pentru a-i solicita o decizie în privința fiecărui executabil. Dacă acesta consideră că un executabil prezintă o amenințare asupra sistemului, acel executabil va fi adăugat într-o listă cu executabile nesigure, care mai departe va fi prelucrată marcând fiecare element din listă cu o regulă fapolicyd care nu permite accesul niciun utilizator la executabilul respectiv.

2.4.2. Algoritm aplicare politici

Toate politicile existente vor fi găsite în folderul politici stocate în fișiere de tipul Json Structura unei politici este următoarea:

```
1       {
2     "name":"nume_politica",
3     "APPS":
4           {
5       "whitelist":[ ],
6       "blacklist":[ ]
7           },
8     "TARGET\_TYPE":"user/group",
9           "TARGET\_NAME":"name-user/name-group"
10      }
```

Listing 2.1. Structura politica

În câmpul `name` se regăsește numele politicii urmat de câmpul APPS care este format din două liste whitelist și blacklist.

`Whitelist` este reprezentat de o listă de path-uri către orice tip de executabil, dar și de director existent în sistemul de operare asupra căruia se dorește marcarea ca obiect sigur, iar utilizatorul să poată accesa această resursă.

`Blacklist` este reprezentat la fel de o listă de path-uri către directoare sau executabile, iar permisiunile asupra acestora urmează să fie restricționate, path-urile urmând să fie marcate ca obiecte nesigure pentru sistem TARGET_TYPE reprezintă tipul de utilizator asupra căruia dorim să-i modificăm drepturile, acesta putând fi de tipul user sau grup de useri iar TARGET_NAME trebuie să fie numele utilizatorului/grupului.

Algoritmul 2.1 Pseudocod pentru funcția de procesare a politicii

function PROCESS_JSON(file)

```
Salvează conținutul fisierului file în p_json
data ← json.loads(p_json)
apps ← data["APPS"]
whitelist ← apps["whitelist"]
blacklist ← apps["blacklist"]
tip ← data["TARGET_TYPE"]
destinatar ← data["TARGET_NAME"]
Golește conținutul fișierului "/etc/fapolicyd/rules.d/allow_politic.rules"
Golește conținutul fișierului "/etc/fapolicyd/rules.d/deny_politic.rules"
Creează reguli pentru blacklist cu create_rules(blacklist, 0, tip,
destinatar)
Creează reguli pentru whitelist cu create_rules(whitelist, 1, tip,
destinatar)
```

Funcția de procesare are următoarele etape de execuție:

- Identificarea politicii trimise în cadrul comenzii;
- Citirea politicii și returnarea acesteia într-o variabilă denumită dată cu ajutorul funcției `loadjson`, funcție care face ca elementele unui json să fie accesibile într-un limbaj de programare ca python;
- Instantierea datelor primite de la politica în obiecte pe care le vom folosi pentru crearea de reguli;
- Funcția `create_rules` cu parametrii:list,verdict,tip și destinatar va crea reguli pentru fiecare obiect din lista primită în funcție de verdict(0=nesigur,1=sigur),tipul politicii(user sau grup) și numele destinatarului. Această funcție va fi apelată de două ori: o dată pentru path-urile sigure cu verdict=1 și o dată pentru path-urile nesigure cu verdict=0.

Capitolul 3. Implementarea aplicației

3.1. Implementarea comunicațiilor Dbus

În realizarea acestei componente, am utilizat un fisier config_file.txt vizibil pentru toate modulele lucrării în care vor fi stocate datele necesare implementării comunicării intre module, conținutul acestui fișier fiind următorul:

- `com.program.interfata`

  Identificator pentru interfața Dbus unde vor fi stocate toate metodele și semnalele implementate.

- `com.program.service`

  Identificator pentru serviciul Dbus care va expune interfața prezentată anterior.

- `/com/program/object`

  Identificator pentru obiectul Dbus. Un obiect de tipul acesta reprezintă o instanță pentru serviciul. `com.program.service` și implementează interfata`com.program.interfata`.

După citirea acestor date din fișier în următoarele variabile: `interface, service și object_address`, am definit clasa `MydbusObject` pentru a expune metodele și semnalele implementate astfel:

```
1   class MyDbusObject(dbus.service.Object):
2
3       @dbus.service.method(interface,signature='s')
4       def emit_semnal_sendJson(self,file):
5           self.sendJson_semnal(file)
6       @dbus.service.signal(interface,signature='s')
7       def sendJson_semnal(self,file):
8           pass
```

Listing 3.1. Exemplu pentru implementarea metodelor si semnalelor

Mai sus se regăsește doar o secvență de cod realizată din implementarea tuturor funcționalităților, restul implementării se va regăsi în Anexa1. Metoda definită mai sus `emit_semnal_sendJson` funcționează ca un declanșator pentru semnalul `sendJson_semnal`, iar atunci când se trimite o politică din modulul de comandă, modulul de execuție va citi semnalul emis prelucrând mai departe datele.

În contiuare, am creat o instanță de tipul `dbus.SessionBus()` căruia i-am atribuit serviciul din configurații. Mai departe am creat un obiect de tipul MyDbusObject pe baza adresei obiectului și a sesiunii dbus. Programul a fost menținut în executie cu ajutorul unei instanțe de tipul `GLib.MainLoop()`. Codul aferent descrierii de mai sus este:

```
1   session_bus=dbus.SessionBus()
2   name=dbus.service.BusName(service,bus=session_bus)
3   object=MyDbusObject(session_bus,object_address)
4   loop=GLib.MainLoop()
5   loop.run()
```

Listing 3.2. Crearea sesiunii Dbus

3.2. Implementarea modulului de executie

În acest modul se va defini un anumit număr de funcții care vor juca rolul unor interceptori de semnale pe canalul dbus și care vor primi ca parametrii, aceleasi variabile ca metodele definite anterior. Conexiunea dintre funcție și metoda va fi realizată cu ajutorul interfeței definite în modulul anterior prin conectarea fiecărei funcții cu câte un semnal din cadrul interfeței.

Codul aferent conexiunilor este:

```
1  def read_json(file):
2      load_json.process_json(file)
3  session_bus=dbus.SessionBus()
4  object=session_bus.get_object(service,object_address)
5  interface=dbus.Interface(object,interface)
6  interface.connect_to_signal('sendJson_semnal',read_json)
7  loop=GLib.MainLoop()
8  loop.run()
```

Listing 3.3. Interceptare activare semnal trimitere politică

Din liniile de cod prezentate, se observă cum este definită și procesată funcția `read_json` care va fi asociată cu semnalul sendJson_Semnal prin intermediul funcției `connect_to_signal`. Atunci când semnalul este activ, funcția `read_json` va prelua datele semnalului și va apela la rândul ei, funcțiile necesare în funcție de comanda primită. Restul conexiunilor se vor găsi la Anexa 2.

3.3. Implementarea modulului de comandă

În acest modul a fost tratată comunicarea directă dintre administrator și componenta Dbus prin transmiterea comenzilor de către acesta. Pentru automatizarea întregului sistem, am utilizat inserarea unei comenzi cu tot cu parametrii prin intermediul liniei de comandă și al argumentelor. Pentru o bună înțelegere a comenzilor și felul în care acestea se apelează, am introdus fișierul help.txt unde am descris aceste funcționalități.

3.4. Implementarea funcționalităților

3.4.1. Procesarea politicilor

Conform pseudopodului prezentat în capitolul 2, în prima etapă am procesat Json-ul trimis ca parametru și am salvat datele acestuia în diferite variabile și mai departe am creat reguli pentru listele de path-uri sigure și suspecte cu ajutorul funcției `create_rules`.

Etapele funcției `create_rules(paths,trusted,tip,destinatar)` sunt:

- Parcurgerea fiecarui element aflat în lista paths;
- Verificarea dacă path-ul trimis este director sau executabil;
- Verifică tipul de politica din variabila `trusted`care poate fi 0 pentru suspect si 1 pentru sigur;
- Verificarea tipului de destinatar care poate fi de 2 tipuri: user și group;
- Verificarea destinatarului, acesta poate fi: numele utilizatorului, numele grupului, sau poate avea valoarea "all" care va cuprinte toți utilizatorii, inclusiv pe administrator ;
- În funcție de datele introduse, se va crea pentru fiecare path o regulă;
- Se vor crea în folderul `/etc/fapolicyd/rules.d` fișierele: `allow_politic.rules` unde se vor stoca regulile pentru path-urile sigure și `deny_politic.rules` unde se vor afla regulile pentru fișierele suspecte.

Un exemplu de politică existent în lucrare este all_users.json, care se adresează tuturor utilizatorilor și care are următoarea structură:

```
1  {
2      "name":"all_users",
3      "APPS":{
4          "whitelist":["/usr/bin/ls","/usr/bin/wget"],
5
   ↪  "blacklist":["/var/tmp","/home/student/f/g.py","/home/student/documents",
6          "/usr/licenta","/usr/games/testare"]
7      },
8      "TARGET_TYPE":"user",
9      "TARGET_NAME":"all"
10 }
```

Listing 3.4. Exemplu politică

Dupa procesarea acestui json, în fișierele menționate mai sus au fost scrise următoarele reguli:

- `allow perm=execute all:  path=/usr/bin/ls`
- `allow perm=execute all:  path=/usr/bin/wget`
- `deny perm=open all:  dir=/var/tmp`
- `deny perm=execute all:  dir=/home/student/f/g.py`
- `deny perm=open all:  dir=/home/student/documents`
- `deby perm=open all:  path=/usr/licenta`
- `deby perm=open all:  path=/usr/games/testare`

Codul pentru funcțiile de procesare se vor regăsi la Anexa3.

3.4.2. Scrierea de reguli pentru executabilele scanate

Conform figurii 2.5, unde ne este prezentată structura acestui proces rezultă că această comandă poate fi apelată doar în urma unei scanări pentru a putea accesa fișierele rezultate.

Etapele acestui algoritm sunt următoarele:

- Citește executabilele din fișierul txt trimis ca parametru
- Trimite câte un executabil către administrator pentru a decide dacă restricționează accesul la resursă sau nu. În urma acestei interogări, se va decide dacă se va crea o regulă pentru executabil sau nu.
- Se returnează o listă cu reguli rezultate în urma interacțiunii anterioare cu adminul
- Se scriu toate regulile primite în fișierul `/etc/fapolicyd/rules.d/deny_scan.rules`, una câte una, deoarece comunicațiile pe dbus se realizează doar prin mesaje și nu se pot trimite obiecte de tipul lista.
- Se aplică un update pentru fapolicyd

Funcțiile pe care le-am utilizat în acest proces sunt următoarele:

```
1   def write_to_fapolicyd(rule,file):
2       lista_reguli=read_exe_from_file(file)
3       lista_reguli.append(rule)
4       all=list(set(lista_reguli))
5       write_list_to_file(all,file)
6
7   def create_rules_for_all_non_root_users(file):
8       exec_untrust=read_exe_from_file(file)
9       list_rules=[]
10      for i in exec_untrust:
11          print(f"\n {i}  Press  1=True  |   0=False")
12          while True:
13              verdict=input()
14              if verdict=='0' or verdict=='1':
15                  break
16          if verdict=='1':
17              list_rules.append(f"deny perm=any all : path={i}")
18      return list_rules
```

Listing 3.5. Interceptare activare semnal trimitere politică

Modul în care au fost apelate aceste funcționalități este acesta:

```
1   list=scanare.create_rules_for_all_non_root_users("./scan_files/"+
2                                       "exe_untrust.txt")
3   for i in list:
4       interface.emit_semnal_send_scan_rule(i)
```

În variabila list vor ajunge toate regulile, iar acestea vor fi trimise pe rând ca mesaje prin intermediul dbus-ului. Metoda `emit_semnal_scan_rule(rule)` va activa semnalul de trimitere, iar funcția `write_to_fapolicyd()` va fi apelată.

3.4.3. Inserarea de reguli

Separat de modulul de scriere al datelor, în fișierele fapolicyd prin aplicarea politicilor și scrierea regulilor în urma scanării, am implementat o funcționalitate care poate să trimită o singură comandă către fapolicyd.

De exemplu :

```
$ ./send_command.py insert_rule /tmp 0 user student
```

Linia de comandă prezentată mai sus, apelează funcția `insert_rule` cu următorii parametrii`(path,verdict,tip_p,destinatar)`, unde path primește valoarea /tmp, verdict primeste 0, `tip_p=user` și destinatar primește valoarea `student` și aplică un algoritm asupra acestor date asemănător cu cel de la crearea de reguli pentru politici. Singura diferență este că regula va fi scrisă în alt fișier de reguli aflat la adresa `/etc/fapolicyd/rules.d/rules_by_user.rules`. Consider că implementarea unei astfel de funcții este strict necesară, deoarece permite restricționarea unei resurse, la alegerea administratorului fără a fi nevoit să apeleze la diferite politici.

3.4.4. Resetarea sistemului

Datorită regulilor predefinite de fapolicyd, care influențează într-un mod sau altul funcționalitatea acestei aplicații am fost nevoit să scriu o funcție care deschide fiecare fișier de reguli și șterge conținutul acestora. Aici vor fi mai multe tipuri de resetări în funcție de parametrul primit în variabila tip.

Codul funcției de reset este următorul:

```
1   def reset_rules(tip):
2       dir='/etc/fapolicyd/rules.d'
3       if(tip=="all_rules"):
4           for file in os.listdir(dir):
5               path=dir+"/"+file
6               with open(path,"w") as fisier:
7                   fisier.truncate()
8
9       elif(tip=="politic_rules"):
10          with open(dir+"/allow_politic.rules","w") as fisier:
11              fisier.truncate()
12
13          with open(dir+"/deny_politic.rules","w") as fisier:
14              fisier.truncate()
15      elif(tip=="scan_rules"):
16          with open(dir+"/deny_scan.rules","w") as fisier:
17              fisier.truncate()
18      elif (tip=="admin_rules"):
19          with open(dir+"/rules_by_user","w") as fisier:
20              fisier.truncate()
21      os.system("systemctl stop fapolicyd")
22      os.system("fagenrules --load")
```

Listing 3.6. Functia de resetare

3.4.5. Comenzi atribuite demonului fapolicyd

Pentru o interacțiune eficientă cu fapolicyd am implementat niște comenzi care au rolul de a porni și opri demonul, dar și o comandă pentru update în care se vor încarcă toate modificările.

Funcțiile care asigură aceste funcționalități sunt:

```
1    def update_fapolicyd():
2       os.system("systemctl stop fapolicyd")
3       os.system("fagenrules --load")
4       os.system("systemctl start fapolicyd")
5
6   def stop_fapolicyd():
7       os.system("systemctl stop fapolicyd")
8
9   def start_fapolicyd():
10      os.system("systemctl start fapolicyd")
```

Listing 3.7. Functiile destinate Fapolicyd

De menționat este că atunci când dorim să încărcam regulile scrise în fișiere cu comanda `fagenrules -load`, trebuie oprit sistemul și după încărcarea regulilor, să-l repornim la loc.

Capitolul 4. Testarea aplicației și rezultate experimentale

4.1. Punerea în funcțiune

Prima etapă din testarea aplicației pe stația noastră de lucru, este verificarea existenței demonul fapolicyd. În cazul în care nu există, se poate instala ușor, folosind comanda:

```
$ yum install fapolicyd
```

Pentru a lansa în execuție aplicația noastră, este nevoie de deschiderea în paralel a trei terminale configurate astfel:

- Pe primul terminal va rula programul `dbus_server.py` prin introducerea comenzii:

  ```
  $ ./dbus_server.py
  ```

- Pe al doilea terminal va rula modulul de execuție si anume `read_command.py` prin comanda `./read_command.py`;

- Al treilea terminal va reprezenta modulul de comandă, unde va avea loc comunicația directă cu administratorul prin trimiterea de comenzi catre Dbus cu ajutorul executabilului `./send_command.py`.

4.2. Testarea comenzilor

4.2.1. Comanda de reset

În cazul în care este prima utilizare după instalarea demonului fapolicyd se recomandă utilizarea comenzii de `reset_system` cu parametrul `tip=all_rules`, astfel încât să se elimine regulile standard care afectează funcționalitatea aplicației. În cele ce urmează, voi aplica funcția de reset asupra regulilor deja existente în fapolicyd. De menționat este faptul că, comanda `fapolicyd-cli -list` afișează toate regulile care rulează pe stația respectivă.

```
[root@localhost licenta]# fapolicyd-cli --list
1. allow perm=open gid=studenti : dir=/usr/licenta/
2. deny perm=execute gid=studenti : path=/blocaj/p.py
3. deny perm=open gid=studenti : dir=/var/tmp
4. deny perm=open gid=studenti : dir=/usr/games/testare
5. deny perm=open gid=studenti : dir=/usr/documents
6. allow perm=execute uid=student : path=/usr/licenta/functii.py
7. deny perm=execute all : path=/usr/licenta/functii.py
[root@localhost licenta]# ./send_command.py reset_system all_rules
[root@localhost licenta]# fapolicyd-cli --list
[root@localhost licenta]# _
```

Figura 4.1. Reset_command

4.2.2. Comanda de aplicare politica

Pentru a testa funcționalitatea implementării politicilor de control pe stațiile de lucru, am ales să utilizez o politică de tipul grup, care are următoarea structură:

```
1       {
2     "name":"studenti"
3     "APPS":{
4     "whitelist":["/usr/licenta/"],
5     "blacklist":["/blocaj/p.py","/var/tmp","/usr/games/testare","/usr/documents"]
6
7     },
8     "TARGET_TYPE":"group",
9     "TARGET_NAME":"studenti"
10      }
```

Listing 4.1. Politica studenti

După rularea comenzii `./send_command.py send_policy studenti.json`, au fost create următoarele reguli:

```
[root@localhost licenta]# fapolicyd-cli --list
[root@localhost licenta]# ./send_command.py send_policy studenti.json
[root@localhost licenta]# fapolicyd-cli --list
1. allow perm=open gid=studenti : dir=/usr/licenta/
2. deny perm=execute gid=studenti : path=/blocaj/p.py
3. deny perm=open gid=studenti : dir=/var/tmp
4. deny perm=open gid=studenti : dir=/usr/games/testare
5. deny perm=open gid=studenti : dir=/usr/documents
[root@localhost licenta]#
```

Pentru verificarea restricționării, mă voi loga cu un user care face parte din grupul de studenți și voi încerca să accesez executabilul `/blocaj/p.py` , și fișierele din `/usr/documents`

```
[student1@localhost ~]$ cd /blocaj
[student1@localhost blocaj]$ ./p.py
-bash: ./p.py: Operation not permitted
[student1@localhost blocaj]$
```

```
[student1@localhost documents]$ ls
file.txt
[student1@localhost documents]$ cp file.txt /home/student1
cp: cannot open 'file.txt' for reading: Operation not permitted
[student1@localhost documents]$ _
```

4.2.3. Comanda de scanare

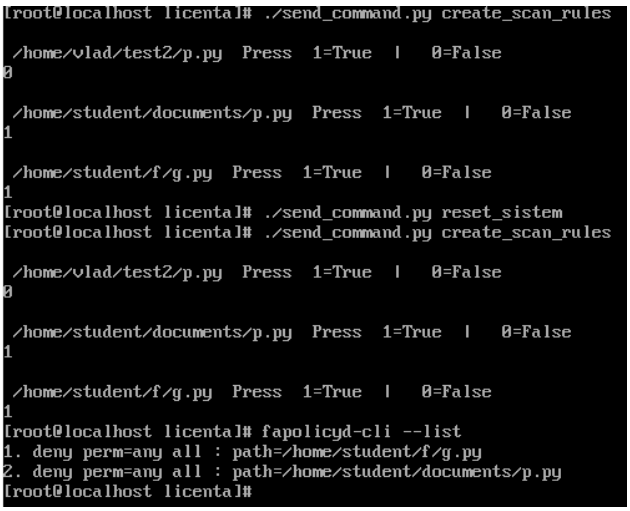
Pentru verificarea funcției de scanare, am utilizat comanda asupra directorului /home pentru a vedea toate executabilele create de utilizatori cu comanda următoare:

`$ ./send_command.py scan_system /home`

În fișierul `exe_untrust`, au fost detectate trei executabile care nu au proveniența sigură și anume:

```
/home/vlad/test2/p.py
/home/student/documents/p.py
/home/student/f/g.py
~
~
```

Mai departe am apelat comanda : `./send_command.py create_scan_rules` și am decis ca primul executabil să-l declar safe și pe restul unsafe. Comunicarea cu sistemul a fost următoarea:



4.2.4. Comanda de insert

Înainte de a rula comanda de insert, am rulat iar comanda de reset pentru o evidențiere mai bună a comenzii.

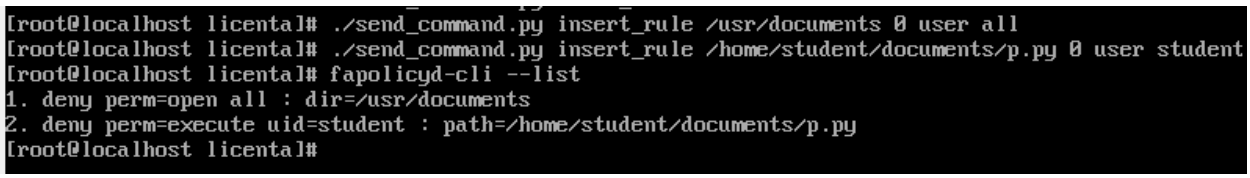
În cazul de față dorim să restricționăm accesul la folderul `/usr/documente` pentru toți utilizatorii, și la executabilul `/home/student/documents/p.py` pentru userul student.

Am inserat comenzile următoare:

`./send_command.py insert_rule /usr/documents 0 user all`

`./send_command.py insert_rule /home/student/documents/p.py 0 user student`

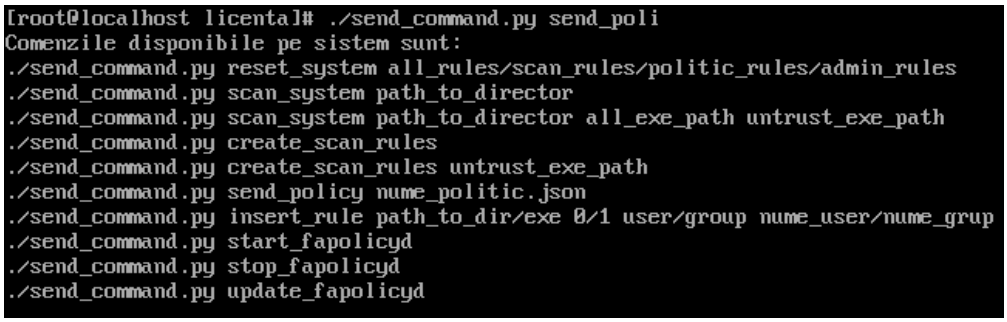
În momentul de față, niciun utilizator fără drepturi de administrator nu va putea accesa niciun fișier sau executabil din folderul `/usr/documente` și user-ul student nu va putea accesa executabilul `/home/student/documents/p.py` aflat în subdirectorul său.



4.2.5. Comanda eronată

În cazul în care se introduce o comandă eronată, care nu este definită în sistem, se va afișa textul dintr-un fișier `help.txt` aflat în folderul cu lucrarea în care se vor descrie toate combinațiile de reguli existente pe sistem.

Un exemplu de rulare ar fi:



Concluzii

În cadrul acestei lucrări, s-a încercat implementarea unui sistem de securitate la nivel de aplicații pentru prevenirea unui atac cibernetic de tipul Malware. Acest lucru s-a materializat prin introducerea unui concept de: "politici bazate pe etichete". Am ales să utilizez acest concept deoarece reprezinta una dintre cele mai des întâlnite soluții pentru securitatea resurselor unui sistem. Mai mult decât atât, a fost implementat un modul de scanare, care îi oferă administratorului opțiunea de a scana întreg sistemul în vederea evitării introducerii unui executabil rău intenționat care poate avea o influenta negativa asupra integrității stației de lucru.

Pentru realizarea lucrării s-a pus accentul pe extinderea demonului Fapolicyd, pentru a-l face capabil să ruleze într-un mediu manipulat direct de către administrator prin inserarea de diferite comenzi, care vor aduce modificări în fișierele de configurare ale acestui demon. Principalele comenzi implementate menite sa trateze obiectivele lucrării sunt:

- Inserarea și aplicarea unei politici de control
- Scanarea sistemului și crearea de reguli pentru executabilele untrust
- Manipularea directa a demonului Fapolicyd

Toate aceste comenzi au fost testate, iar impactul acestora asupra sistemului de securitate a fost unul pozitiv, aducând astfel un strat superior de protecție împotriva unor atacuri cibernetice.

Direcții viitoare de dezvoltare

Una dintre posibilele direcții de dezvoltare ar fi introducerea acestei lucrări într-un sistem mai complex care sa implementeze la fel conceptul de "securitate bazata pe politici de control" pe diferite arii de securitate. Implementarea unui astfel de sistem ar putea fi utilizată de exemplu într-un cadru universitar pentru susținerea de examene sau teste unde este necesară restricționarea resurselor.

Lecții învățate pe parcursul dezvoltării lucrării de diplomă

Printre lecțiile învățate în cadrul acestei lucrări se numără:

- Datorită funcționalităților implementate am învățat și deprins diferite calități în urma programării în Python.
- Pentru realizarea comunicației dintre module am învățat modul de utilizare și configurare al comunicațiilor Dbus.
- Am învățat modul de utilizare al demonului Fapolicyd pentru a-l putea utiliza în licență.
- Datorita implementării acestei lucrări într-un server de CentOS8, am fost nevoit sa ma adaptez la un mediu de lucru unde principala cale de comunicare cu sistemul este linia de comanda.

Bibliografie

[1] “Ce este un atac cibernetic.” [Online]. Available: https://www.microsoft.com/ro-ro/security/business/security-101/what-is-a-cyberattack

[2] “Malware.” [Online]. Available: https://www.microsoft.com/ro-ro/security/business/security-101/what-is-malware

[3] “How fapolicyd works:.” [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/assembly_blocking-and-allowing-applications-using-fapolicyd_security-hardening

[4] “Phishing.” [Online]. Available: https://www.microsoft.com/ro-ro/security/business/security-101/what-is-phishing

[5] “How selinux works.” [Online]. Available: https://phoenixnap.com/kb/selinux

[6] “How apparmor works.” [Online]. Available: https://phoenixnap.com/kb/apparmor-vs-selinux

[7] “How smack works.” [Online]. Available: https://www.kernel.org/doc/html/v4.18/admin-guide/LSM/Smack.html

[8] “How to create fapolicyd rules.” [Online]. Available: https://www.mankier.com/5/fapolicyd.rules

[9] “Fapolicyd-commands.” [Online]. Available: https://novalug.org/docs/fapolicyd.thurston.pdf

[10] “Comunicatiile pe dbus.” [Online]. Available: https://www.wikiwand.com/en/D-Bus/

[11] “How systemd works.” [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd

[12] “Find command.” [Online]. Available: https://ro.admininfo.info/c-mo-usar-comando-find-en-linux

[13] “Rpm command.” [Online]. Available: https://ro.eyewated.com/rpm-comanda-linux-comanda-unix/

Anexe

Anexa 1. Codul componentei dbus

```
1  import dbus
2  import dbus.service
3  from dbus.mainloop.glib import DBusGMainLoop
4  from gi.repository import GLib
5  DBusGMainLoop(set_as_default=True)
6  with open("/root/test/licenta/config_file.txt",'r')as file:
7      configs=[linie.strip() for linie in file.readlines()]
8
9  interface=configs[0]
10 service=configs[1]
11 object_address=configs[2]
12 class MyDbusObject(dbus.service.Object):
13
14     @dbus.service.method(interface,signature='s')
15     def emit_semnal_sendJson(self,file):
16         self.sendJson_semnal(file)
17     @dbus.service.signal(interface,signature='s')
18     def sendJson_semnal(self,file):
19         pass
20
21     @dbus.service.method(interface,signature='sss')
22     def emit_semnal_scan(self,dir,file1,file2):
23         self.scan_semnal(dir,file1,file2)
24     @dbus.service.signal(interface,signature='sss')
25     def scan_semnal(self,dir,file1,file2):
26         pass
27
28     @dbus.service.method(interface,signature='s')
29     def emit_semnal_reset(self,tip):
30         self.reset_semnal(tip)
31     @dbus.service.signal(interface,signature='s')
32     def reset_semnal(self,tip):
33         pass
34
35     @dbus.service.method(interface,signature='ssss')
36     def
        ↪ emit_semnal_insert_rule(self,path,verdict,tip_politica,destinatar):
37         self.insert_rule_semnal(path,verdict,tip_politica,destinatar)
38     @dbus.service.signal(interface,signature='ssss')
39     def
        ↪ insert_rule_semnal(self,path,verdict,tip_politica,destinatar):
40         pass
41
42     @dbus.service.method(interface,signature='s')
43     def emit_semnal_send_scan_rule(self,file):
44         self.send_scan_rule_semnal(file)
45     @dbus.service.signal(interface,signature='s')
46     def send_scan_rule_semnal(self,file):
```

```
47            pass
48
49        @dbus.service.method(interface,signature='')
50        def emit_semnal_start_fapolicyd(self):
51            self.start_fapolicyd_semnal()
52        @dbus.service.signal(interface,signature='')
53        def start_fapolicyd_semnal(self):
54            pass
55        @dbus.service.method(interface,signature='')
56        def emit_semnal_stop_fapolicyd(self):
57            self.stop_fapolicyd_semnal()
58        @dbus.service.signal(interface,signature='')
59        def stop_fapolicyd_semnal(self):
60            pass
61        @dbus.service.method(interface,signature='')
62        def emit_semnal_update_fapolicyd(self):
63            self.update_fapolicyd_semnal()
64        @dbus.service.signal(interface,signature='')
65        def update_fapolicyd_semnal(self):
66            pass
67    session_bus=dbus.SessionBus()
68    name=dbus.service.BusName(service,bus=session_bus)
69    object=MyDbusObject(session_bus,object_address)
70    loop=GLib.MainLoop()
71    loop.run()
```

Listing 2. Codul componentei Dbus

Anexa 2. Codul Modulului de Executie

```
1  import dbus
2  from scan_files import scanare
3  from json_files import load_json
4  import functii
5  from dbus.mainloop.glib import DBusGMainLoop
6  from gi.repository import GLib
7  DBusGMainLoop(set_as_default=True)
8  with open("config_file.txt",'r')as file:
9      configs=[linie.strip() for linie in file.readlines()]
10
11 interface=configs[0]
12 service=configs[1]
13 object_address=configs[2]
14
15 def read_scan(dir,file1,file2):
16     print(dir +" "+file1+" "+file2)
17     scanare.scanare(dir,file1,file2)
18 def read_json(file):
19     load_json.process_json(file)
20 def reset_f(tip):
21     functii.reset_rules(tip)
22 def insert_one_rule(path,verdict,tip_politica,destinatar):
23     print("a fost inserata o regula")
24     functii.insert_rule(path,verdict,tip_politica,destinatar)
25 def write_rule_to_fapolicyd(file):
26
       ↪  scanare.write_to_fapolicyd(file,"/etc/fapolicyd/rules.d/deny_scan.rules")
27 def start():
28     functii.start_fapolicyd()
29 def stop():
30     functii.stop_fapolicyd()
31 def update():
32     functii.update_fapolicyd()
33 session_bus=dbus.SessionBus()
34 object=session_bus.get_object(service,object_address)
35 interface=dbus.Interface(object,interface)
36
37 interface.connect_to_signal('scan_semnal',read_scan)
38 interface.connect_to_signal('sendJson_semnal',read_json)
39 interface.connect_to_signal('reset_semnal',reset_f)
40 interface.connect_to_signal('insert_rule_semnal',insert_one_rule)
41 interface.connect_to_signal('send_scan_rule_semnal',write_rule_to_fapolicyd)
42 interface.connect_to_signal('start_fapolicyd_semnal',start)
43 interface.connect_to_signal('stop_fapolicyd_semnal',stop)
44 interface.connect_to_signal('update_fapolicyd_semnal',update)
45 loop=GLib.MainLoop()
46 loop.run()
```

Listing 3. Codul modulului de Executie

Anexa 3. Codul pentru procesarea Politicilor

```
1   import os
2   import json
3   def create_rules(paths,trusted,tip,destinatar):
4       for path in paths:
5           if os.path.isdir(path):
6               if trusted==1 and tip=="user" and destinatar!="all":
7                   with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_trust:
8                       fisier_trust.write(f"allow perm=open
                         ↪  uid={destinatar} : dir={path}\n");
9               elif trusted==1 and tip=="group":
10                  with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_trust:
11                      fisier_trust.write(f"allow perm=open
                         ↪  gid={destinatar} : dir={path}\n");
12              elif trusted==0 and tip=="user" and destinatar!="all":
13                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
14                      fisier_untrust.write(f"deny perm=open
                         ↪  uid={destinatar} : dir={path}\n");
15              elif trusted==0 and tip=="group":
16                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
17                      fisier_untrust.write(f"deny perm=open
                         ↪  gid={destinatar} : dir={path}\n");
18              elif trusted==0 and tip=="user" and destinatar=="all":
19                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
20                      fisier_untrust.write(f"deny perm=open all :
                         ↪  dir={path}\n");
21              elif trusted==1 and tip=="user" and destinatar=="all":
22                  with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_untrust:
23                      fisier_untrust.write(f"allow perm=open all :
                         ↪  dir={path}\n");
24
25          elif os.path.isfile(path) and os.access(path,os.X_OK):
26              if trusted==1 and tip=="user" and destinatar!="all":
27                  with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_trust:
28                      fisier_trust.write(f"allow perm=execute
                         ↪  uid={destinatar} : path={path}\n");
29              elif trusted==1 and tip=="group":
```

```
30                  with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_trust:
31                      fisier_trust.write(f"allow perm=execute
                         ↪  gid={destinatar} : path={path}\n");
32              elif trusted==0 and tip=="user"and destinatar!="all":
33                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
34                      fisier_untrust.write(f"deny perm=execute
                         ↪  uid={destinatar} : path={path}\n");
35              elif trusted==0 and tip=="group":
36                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
37                      fisier_untrust.write(f"deny perm=execute
                         ↪  gid={destinatar} : path={path}\n");
38              elif trusted==1 and tip=="user" and destinatar=="all":
39                  with open
                     ↪  ("/etc/fapolicyd/rules.d/allow_politic.rules","a")
                     ↪  as fisier_trust:
40                      fisier_trust.write(f"allow perm=execute all :
                         ↪  path={path}\n");
41              elif trusted==0 and tip=="user"and destinatar=="all":
42                  with open
                     ↪  ("/etc/fapolicyd/rules.d/deny_politic.rules","a")
                     ↪  as fisier_untrust:
43                      fisier_untrust.write(f"deny perm=execute all :
                         ↪  path={path}\n");
44
45  def process_json(file):
46      path=f"./politici/{file}"
47
48      with open(path) as fisier:
49          p_json=fisier.read()
50
51      data=json.loads(p_json)
52
53      apps=data["APPS"]
54      whitelist=apps["whitelist"]
55      blacklist=apps["blacklist"]
56      tip=data["TARGET_TYPE"];
57      destinatar=data["TARGET_NAME"];
58      with open ("/etc/fapolicyd/rules.d/allow_politic.rules","w") as
         ↪  fisier_trust:
59          fisier_trust.truncate()
60      with open ("/etc/fapolicyd/rules.d/deny_politic.rules","w") as
         ↪  fisier_untrust:
61          fisier_untrust.truncate()
62      create_rules(blacklist,0,tip,destinatar)
63      create_rules(whitelist,1,tip,destinatar)
```

Listing 4. Codul pentru procesarea Politicilor

Anexa 4. Structura de fisiere:

In urma modulelor create structura lucrării va fi urmatoarea: