Laboratorul nr. 2
Instrumente pentru asigurarea unei securități sporite la nivel de stație de lucru în Linux
protecție prin virtualizare

Se va utiliza o mașină virtuală cu un debian bullseye proaspăt instalat

1. AppArmor

Pentru instalare sub Debian

```
apt install apparmor apparmor-utils auditd
apt install apparmor-profiles apparmor-profiles-extra
mkdir -p /etc/default/grub.d
```

Creați apoi fișierul /etc/default/grub.d/apparmor.cfg cu următorul conținut.
GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT apparmor=1 security=apparmor"
După ce se salvează acest fișier se va executa

```
update-grub
```

Este un instrument din categoria celor care asigură întărirea controlului accesului (mandatory access control sau MAC) pentru Linux. Este proiectată să utilizeze controlul discreționar la acces standard (DAC) al permisiunilor din UNIX. Principalul ei scop este că permite crearea de fișiere de configurare a accesului de orice tip pentru oricare aplicație din sistem. Distribuțiile serioase furnizează niște fișiere implicite care pentru un utilizator simplu sunt satisfăcătoare dar de multe ori nici nu există. Comanda este de nivel supervizor. Eu lucrez în consolă supervizor când am administrare de făcut deci pentru cei care nu consideră necesar acestu lucru exemplele trebuie precedate de sudo.

Din nefericire orice utilizator sau administrator de rețea trebuie să aibă cunoștințe despre această configurare manuală particularizată datorită existenței următoarelor cazuri:

- Când sunt prea multe aplicații care trebuie să interopereze este posibil ca:
  1. Unele sa aibă profil iar altele să nu fie deloc supravegheate
  2. Profilul implict furnizat de distribuție sau de aplicației la instalare să aibă unele reguli care blochează parțial interoperarea (destul de greu de detectat deoarece de multe ori se manifestă aleatoriu acest blocaj (nu pe funcționalitățile dominante) deci trebuie analizat înainte de instalare această situației
- Când se folosesc simultan mai multe sisteme de asigurare a securității pe stație se poate ca acestea să se încurce reciproc (de exemplu - firewall linux cu apparmour cu firejail)

Din acestea rezultă clar că cunoașterea tuturor metodelor de configurare pentru orice aplicație care asigură securitatea la un anumit nivel este critică pentru un utilizator de Linux. Aceasta este de fapt una din problemele care apar când utilizatorii de Windows obișnuiți cu aplicații cu un nivel mare de configurare automată decid sau încearcă să treacă pe Linux din motive evidente la ora actuală.

După cum era de așteptat primul lucru care trebuie făcut este să se verifice dacă aplicația este instalată și activă. Acest lucru se realizează cu următoarele comenzi:

systemctl status apparmor cu o porțiune din rezultat

```
● apparmor.service - Load AppArmor profiles
   Loaded: loaded (/lib/systemd/system/apparmor.service; enabled; vendor preset:
   Active: active (exited) since Wed 2019-07-17 08:49:55 EEST; 1h 34min ago
     Docs: man:apparmor(7)
```

Pentru situația în care aceasta nu este pornită avem următoarele posibilități:

```
systemctl start apparmor    # care pornește serviciul și
systemctl enable apparmor   # o configurează pentru pornirea automată la boot
```

Odată verificat faptul că aplicația este activă putem utiliza aa-tools pentru monitorizarea și gestiunea sistemului. Pentru început vom folosi această comandă pentru a verifica ce profile de aplicații sunt încărcate precum și starea lor curentă.

```
aa-status
apparmor module is loaded.
47 profiles are loaded.
```

```
45 profiles are in enforce mode.
   /snap/core/7270/usr/lib/snapd/snap-confine
   /snap/core/7270/usr/lib/snapd/snap-confine//mount-namespace-capture-helper
   ...
2 profiles are in complain mode.
   libreoffice-oopslash
   libreoffice-soffice
8 processes have profiles defined.
8 processes are in enforce mode.
   /usr/sbin/cups-browsed (878)
   ...
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

Pentru acest caz se observă că în general situația este bună dar profilele de libreoffice sunt în complain mode. Aceasta se datorează faptului că am eliminat complet aplicația neglijând să șterg profilele asociate din apparmor. Faptul că servicii din cups (și nu numai) sunt în enforced mode este benefic știut fiind că acestea prezintă un risc major de securitate.
Atunci când un profil se află în mod complain apparor va permite totuși execuția majorității task-urilor fără restricții (mai puțin cele impuse explicit cu deny în profilul ei) dar cu înregistrarea lor în jurnalul de audit ca evenimente.

Modul unconfined trebuie evitat deoarece aplicația va fi executată fără verificări dar și fără jurnalizarea execuției (nu este deloc sub controlul apparmor)
Comanda nu va afișa informații despre aplicațiile care nu au un profil asociat!

După cum am discutat până acum există multe situații în care utilizatorul trebuie să creeze profile personalizate pentru o aplicație. Pentru simplitate vom exemplifica utilizarea comenzii aa-genprof prin crearea unui script simplu și apoi a unui profil pentru el.

Dacă doresc să văd programele care sunt în modul confined vom da:

```
ps auxZ | grep -v '^unconfined'
```

Se va crea undeva un director testapparmor unde vom crea respectivul script apoi în interiorul lui un subdirector data deoarece profilul exemplu va urmări interzicere accesului într-un anumit director pentru respectivul script. Acum să trecem la crearea scriptului test.sh. Eu prefer nano dar este bun orice alt editor în mod consolă sau nu. Evident că la nivel de supervizor este recomandată evitarea utilizării aplicațiilor grafice.

```
root@home:~# mkdir testapparmor
root@home:~# cd testapparmor/
root@home:~/testapparmor# mkdir data
root@home:~/testapparmor# nano test.sh
root@home:~/testapparmor# chmod a+x test.sh
```

Scriptul va conține următoarele comenzi:

```
#!/bin/bash
echo "Testare apparmur."
touch data/sample.txt
echo "Am creat fisierul"
rm data/sample.txt
echo "Am sters fisierul"
```

Apoi testăm respectivul script:

```
root@home:~/testapparmor# ./test.sh
Testare apparmur.
```

Am creat fisierul
Am sters fisierul

Acum putem trece la crearea unui profil pentru apparmor. Există două comenzi care pot fi folosite în acest scop aa-genprof și aa-logprof. Prima este utilizată pentru a monitoriza și crea pentru prima oară un profil al unei aplicații și are un dialog oferit de apparmor după ce analizează aplicația care îi permite acesteia să stabilească ce i se permite sau nu în timpul execuției respectivei aplicații. Cea de a doua se utilizează pentru un profil existent și permite modificarea sau adăugarea unor reguli în urma analizei aplicației și în special pentru cazurile în care profilul se află în modul enforce sau comply. Pentru a le putea utiliza trebuie ca să fie instalat pachetul de utilitare a lui apparmor:

apt install apparmor-utils

În continuare **se va deschide un nou terminal** în care vom executa scriptul. După cum am spus pentru această configurare inițială aparmor va lansa în execuție și va monitoriza fiecare eveniment generat de aplicație și apoi va propune reguli cu privire la maniera de gestionare particularizată. La lansare comanda va avertiza că pentru a evita suprascrierea unui profil existent ar trebui verificat dacă acesta nu este deja creat (caz în care trebuie folosită cea de a doua comandă). După ce se va executa scriptul trebuie apăsata tasta s.

```
root@home:~# cd testapparmor/
root@home:~/testapparmor# aa-genprof test.sh
Writing updated profile for /root/testapparmor/test.sh.
Setting /root/testapparmor/test.sh to complain mode.
...
Profiling: /root/testapparmor/test.sh
...
[(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
```

Acum vom deschide o nouă consolă în care vom executa scriptul. Apoi ne întoarcem în terminalul doi și apăsăm s. Va apărea următorul rezultat:

```
Profile:  /root/testapparmor/test.sh
Execute:  /bin/touch
Severity: unknown
(I)nherit / (C)hild / (N)amed / (X) ix On / (D)eny / Abo(r)t / (F)inish
```

În general după analiza execuției apparmor urmărește dacă aplicația testat încearcă accesarea sau execuția unor fișere sau dacă cere un anumit nivel de acces la kernel. Totodată ni se furnizează un nivel de severitate pentru respectiva operație (necunoscut sau între 1 și 10-cel mai mare risc). Totuși utilizatorii cu experientă pot decide ignorarea avertismentelor în condițiile unor justificări ferme (de genul: nu este așa de grav sau accept scăderea nivelului de securitate pentru că mai am măsuri conexe sau nu am altă opțiune ca să îmi meargă corect o interoperabilitate și nu am o politică ferm stabilită care îmi interzice respectiva decizie). Un exemplu simplu (nu că aș considera Chromium sigur vreodată) este situația în care în cazul analizei se raportează faptul că Chromiul solicită acces la CAP_SYS_ADMIN pentru a crea un sandbox sigur pentru procesele copil dar apparmor ne dă o severitate 10 pentru această operație

Ce înseamnă fiecare opțiune prezentată de către apparmor în procesul de creare primară a profilului.

- **I**nherit: crează în profil o regulă marcată cu “ix” care permite unui executabil să moștenească automat permisiunile din profilul părinte.
- **C**hild: crează în profil o regulă marcată cu “Cx” care necesită crearea unui profil (în cadrul profilului pentru părinte) cu reguli separate pentru acest copil ceea ce înseamnă cereri de informații suplimentare de la apparmor în momentul execuției respectivului copil.
- **D**eny: crează în profil o regulă marcată cu “deny” la începutul liniei din profil și conduce la intezicerea acesului părintelui la resursa respectivă
- **A**bort: Termină forțat execuția modului de creare profil fără a salva nici o modificare.

- **F**inish: Părăsește execuția curentă dar va salva toate modificările deja efectuate.

Datorită acestor nuanțe este preferabil ca pe cât posibil să se pornească de la un profil existent. În cazul exemplului nostru se poate da permisiune de tip Inherit pentru comenzile touch și mai apoi pentru rm deoarece știm ce este cu ele. Însă procesul de auditare continuă și ne apare următoarea problemă:

```
Profile:    /root/testapparmor/test.sh
Path:       /dev/tty
New Mode: owner rw
Severity: 9
 [1 - #include <abstractions/consoles>]
  2 - #include <abstractions/lightdm>
  3 - owner /dev/tty rw,
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Audi(t) / (O)wner permissions off / Abo(r)t / (F)inish
```

Se observă că în raportare apare un nou câmp numit ”Mode„ care ne precizează ce permisiuni sunt necesare scriptului pentru situația în care acesta accesează o anumită cale. În acest conetxt apar noi opțiuni din care cele mai importante sunt următoarele:

- Allow: Permite accesarea căii cu permisiunile cerute de aplicație
- **D**eny: Interzice accesarea căii utilizând permisiunile cerute
- **I**gnore: Se sare peste această etapă de configurare dar întrebarea va reapărea la următoarea execuție a lui logprof

Deoarece respectivul script tipărește mesaje către consolă el va avea nevoie de acces către interfața TTY și atunci acest lucru poate fi permis. Apoi urmează o întrebare legată de /etc/ld.so.preload care este utilizat pentru a enumera obiectele obiectelor comune ale ELF care trebuie încărcate înainte de execuția unui program. Rezultă că și acestuia i se poate permite execuția (Pe Buster este mai evouluată analiza și nu mai cere o astfel de permisiune).

În următoarea întrebare am ajuns la cererea de acces de tip scriere pentru un fișier aflat în subdirectorul data.

```
Profile:    /root/testapparmor/test.sh
Path:       /root/testapparmor/data/sample.txt
New Mode: owner w
Severity: unknown
```

Deoarece știm ce am cerut în script considerăm această cerere legitimă și o acceptăm. Deoarece procesul de analiză s-a terminat apparmor ne comunică acest lucru și ne întreabă dacă dorim să salvăm modificările. Deci vom apăsa s și apoi f.

În continuare vom trece la modificarea profilului existent pentru o aplicație (în cazul nostru cel asociat script-ului de test)

Deși în principiu ar putea exista idea (comodă dar nefezabilă) că odată creat profilul unei aplicații acesta nu mai trebuie schimbat din nefericire acest lucru nu este valabil. Exemplul tipic este dat de update-uri continuii ale aplicațiilor unde există situații în care cei care produc respectiva aplicație neglijează să furnizeze și un nou profil modificat corespunzător sau mai rău: îl furnizează se suprascrie iar eu din diverse motive am făcut modificări suplimentare pentru profilul curent care evident se pierd.

Pentru a rezolva astfel de situații există două comenzi și anume aa-logprof și aa-mergeprof.
Înainte de a începe modificarea profilului trebuie să fim siguri că acesta a fost creat și încărcat deci se va face aceeași verificare ca în prima situație.

```
aa-status
apparmor module is loaded.
```

50 profiles are loaded.
46 profiles are in enforce mode.
/root/testapparmor/test.sh
/snap/core/7270/usr/lib/snapd/snap-confine
...

Acum putem testa aa-logprof care este utilizat pentru căutarea în jurnalul de auditare după orice evenimente pentru care apparmor nu a reușit să aplice regulile deja prevăzute în profilul aplicației. Pentru aceasta trebuie să generăm astfel de evenimente. În cazul scriptului nostru aceasta se poate face prin modificarea lui în sensul că nu va ma mai scrie în subdirectorul data (care este deja avizat în profilul curent al aplicației).

După modificarea scriptului la o nouă execuție (după cum era de așteptat) apparmor blochează pe tăcute accesul în afara zonelor prestabilite inițial.

```
root@home:~/testapparmor# ./test.sh
Testare apparmur.
touch: cannot touch 'sample.txt': Permission denied
Am creat fisierul
rm: cannot remove 'sample.txt': No such file or directory
Am sters fisierul
```

În urma execuției lui aa-logprof se obține:

```
aa-logprof
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:
Enforce-mode changes:

Profile:    /root/testapparmor/test.sh
Path:       /root/testapparmor/sample.txt
New Mode: owner w
Severity: unknown

 [1 - owner /root/testapparmor/sample.txt w,]
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Audi(t) / (O)wner permissions off / Abo(r)t / (F)inish
```

Se observă că s-au prezentat opțiuni numai pentru acea aplicație care a depășit regulile din profil (în cazul nostru scriptul test) evident că îi vom da allow și mai apoi save și încercăm să reexecutăm scriptul care de această dată va funcționa conform așteptărilor.

Să presupunem că suntem într-un caz în care am două sisteme S1 și S2 aparținând utilizatorilor U1 și U2. Când U1 modifică scriptul de pe S1 va efectua și modificările corespunzătoare în profilul aplicației. U2 face și el în același timp alte schimbări în scriptul de pe S2 și modifică corespunzător și profilul aplicației. La un moment dat U1 și U2 își combină cele două script-uri într-unul singur și ar trebui să facă același lucru cu cele două profile (diferite) pentru AppArmor. Acest lucru este dificil pentru script-uri mari (sunt și de un megabyte). Pentru a rezolva în mod automat această situație există aa-mergeprof.

Să inspectăm profilul curent al aplicației noastre. Toate profilele sunt păstrate în /etc/apparmor.d iar numele profilului este o combinație utilizând punctul ca separator dintre calea și numele aplicației pentru care este destinat acel profil. Rezultă că în cazul nostru numele va fi:

root.testapparmor.test.sh

Dacă îl deschidem într-un editor vom observa următoarele.

```
# Last Modified: Wed Jul 17 13:49:58 2019
#include <tunables/global>
/root/testapparmor/test.sh { //se specifică calea către aplicație
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles> //se preiau reguli generale din bibliotecile referite
  /bin/bash ix,
  /bin/rm mrix,
  /bin/touch mrix,
  /root/testapparmor/test.sh r,
  owner /root/testapparmor/data/sample.txt w,
  owner /root/testapparmor/sample.txt w, // adăugați un deny în față, salvați fișerul
}
```

După modificarea manuală direct pe profil cu un editor noile reguli trebuie reîncărcate. Pentru aceasta dați următoarea comandă:

```
apparmor_parser -r /etc/apparmor.d/home.user.bin.example.sh
```

Când se va relua executarea scriptului acesta va genera din nou aceleași erori ca atunci când regula respectivă nu exista.

Dacă se dorește dezactivarea unui profil anume pentru o aplicație se va da (ex pentru profilul ping)

```
aa-complain /usr/bin/ping    sau aa-disable /path/to/program
```

Pentru a aplica un set de reguli în modul enforced după modificarea profilului aplicației se va da

```
aa-enforce /path/to/program
```

Firejail

Deși există o serie de posibile alternative cum ar fi docker (pe aproape mai ales că la ultimele versiuni suportă și lucrul cu namespace-uri dar procesul părinte se execută la nivel de supervizor ceea ce îți bagă mortul în casă) sau bublewrap (cam limitativ și fără prea mare flexibilitate în configurări) această abordare a fost gândită din start să coopereze cu sistemele native de securitate ale Linuxului inclusiv cu apparmor.

Ea este o aplicație bazată pe conceptul de execuție virtualizată - sandbox care de fapt reprezintă o execuție într-un container.

Pentru a lămuri maniera de funcționare trebuie mai întâi să discutăm despre namespace-urile specifice Linuxului. Acestea reprezintă un concept care permite vizualizarea separată a diverselor părți din sistem. De exemplu dacă două procese se execută în același namespace atunci vor fi capabile să acceseze același fișier sau director și nu vor avea acces la fișierele sau directoarelor care aparțin altor namespace-uri.

În Linux există șapte namespace-uri diferite cum ar fi cel pentru fișiere, cel pentru rețea, cel pentru utilizator, cel pentru grupurile pentru controlul accesului, cele pentru gestiunea numerelor de identificare ale proceselor (PID) și lista poate continua. De exemplu se poate să existe două procese care utilizează aceași interfață de rețea fără a fi capabile să se vadă unul pe altul (namespace diferite deși de același tip adică rețea) sau să existe diverse view-uri aupra unui arbore de procese sau identificatori ale utilizatorilor în funcție de namespace-ul din care se face parte. Pentru a vedea ce namespace-uri sunt disponibile în sistemul nostru se poate da comanda

```
ls /proc/self/ns
cgroup ipc   mnt   net   pid   pid_for_children   user   uts
```

Pentru a inspecta namespace-urile utilizate la un moment dat se poate utiliza

```
lsns
        NS TYPE   NPROCS   PID USER             COMMAND
4026531835 cgroup    238     1 root             /sbin/init
4026531836 pid       231     1 root             /sbin/init
4026531837 user      231     1 root             /sbin/init
4026531838 uts       230     1 root             /sbin/init
4026531839 ipc       238     1 root             /sbin/init
4026531840 mnt       219     1 root             /sbin/init
4026531860 mnt         1    50 root             kdevtmpfs
...
```

Iar pentru a vedea procesele cu drepturi crescute se poate utiliza pscap. Desigur înainte trebuie instalate:

```
apt install libcap-ng-utils
```

Și după lansare:

```
pscap
ppid  pid   name          command              capabilities
1     317   root          systemd-journal      chown, dac_override, dac_read_search, fowner,
setgid, setuid, sys_ptrace, sys_admin, audit_control, mac_override, syslog, audit_read
1     343   root          systemd-udevd        full
1     725   systemd-timesync  systemd-timesyn  sys_time
1     806   root          udisksd              full
1     812   root          cron                 full
1     814   root          rsyslogd             full
...
```

Evident că se poate utiliza grep pentru a extrage informațiile numai despre anumite procese.

Ce ne oferă Linuxul în termeni de asigurare a securității?

Dintotdeauna Linux (ca și bunicul lui UNIX) a pornit de la conceptul existenței unui utilizator de bază (root) cu privilegii complete în gestionarea la orice nivel de acces al sistemului. Dea asemenea are și conceptul de rights elevation (existent și la ultimele versiuni de Windows) care se manifestă prin acea permisiune specială pentru fișiere (Set owner User ID up on execution- SUID) ce permite execuția cu drepturi similare cu ale utilizatorului original chiar dacă acestea sunt mai mari decât ale utilizatorului care are permisiunea de lansare în execuție a ei. Chiar dacă există o logică bine definită în această abordare totodată în contextul conectării continui la diverse rețele prin intermediul unor puncte de acces filtrate parțial sau de loc această abordare a devenit una dintre cel mai vechi probleme de securitate odată cu apariția Internetului. De exemplu un simplu ping are nevoie de acces total la raw sockets pentru a trimite pachete de tip ICMP, adică la un nivel mic pe diagrama ISO-OSI ceea ce ridică mari probleme în cazul unui atac/compromitere a aplicației chiar dacă în principiu se execută în spațiul utilizatorului.

Totuși Linux ne permite ca să stabilim explicit o serie de restricții particularizate asupra ceea ce poate face un utilizator din categoria root. De exemplu operații ca schimbarea deținătorului uni fișier (chown), ignorarea permisiunilor DAC, schimbarea configurației rețelei sau oprirea forțată a unor procese pot fi specificate explicit pentru orice proces se dorește. Totalitatea setărilor disponibile în acest scop poate fi găsită la http://man7.org/linux/man-pages/man7/capabilities.7.html

De asemenea putem inspecta permisiunile pe care le are un anumit fișier ca în exemplul de mai jos:

```
getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

În acest context abordarea bazată pe utilizarea unor sandbox-uri ne crește nivelul de control cu privire la posibilitatea de asociere sau revocare de privilegii pentru orice proces.

Deși la o primă vedere ar trebui ca procesele utilizatorului să aibă o interacțiune cât mai redusă cu kernelul sistemului de operare multe dintre ele nu respectă această regulă.

Programele din spațiul utilizatorului accesează kernelul prin intermediul apelurilor sistem. Sunt mai mult de 300 de astfel de apeluri sistem disponibile în Linux (http://man7.org/linux/man-pages/man2/syscalls.2.html). Tocmai datorită faptului că numai o mică parte din acestea sunt real necesare pentru buna desfășurare a activității unui utilizator curent ele sunt speculate de atacatori prin utilizare de bug-uri din kernel sau o creștere nepermisă de privilegii. Pentru a mai diminua impactul acestor aspecte kernel-ul utilizează *seccomp-bpf* care este o facilitate (https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html) ce permite utilizarea filtrelor Berkeley - BPF (de nivel inferior arhitectural) pentru a defini mai clar lista apelurilor sistem care sunt permise pentru un proces sau un set de procese. UN exemplu simplu este dat de întrebarea dacă nivelul meu de utilizare al uni procesor de texte necesită ca acesta să aibă acces la rețeaua de comunicații a sistemului.

Pentru a evita o suită de configurări inutil (greoaie și de multe ori nu foarte justificate de nivelul posibil de atac la care va fi expus sistemul) Firejail ne furnizează o încapsulare într-un sandbox și este prevăzută și cu foarte multe profile predefinite. Pentru cine este curios să verifice gradul de acoperire cu profile predefinite oferit de firejail este suficient să lanseze următoarea comandă:

```
ls /etc/firejail/
```

Având în vedere numărul mare de profile oferite s-a prevăzut o comandă

```
firecfg
```

care permite configurarea automată a firejail-ului simultan pentru toate aplicațiile pentru care există profile. Aceasta va crea de fapt legături virtuale pentru programele existente în directorul de profile asigurând astfel deschiderea lor automată de către firejail. Dacă se dorește ca firejail să execute o altă aplicație care nu are un profil particularizat (ea oricum are un set de profile generice) atunci se poate crea explicit o legătură simbolică ca mai jos.

```
ln -s /usr/bin/firejail /usr/local/bin/<program name>
```

În situația inversă în care doresc să execut o aplicație fără a mai fi intermediată de către firejail cu o simplă comandă pot afla unde se află executabilul original deoarece știm unde se află legăturile virtuale utilizate pentru lansarea prin intermediul lui firejail

```
which -a nume app
```

Ca mai apoi să o lansăm explicit utilizând calea directă către executabilul original.
Pentru situații de genul navigare privată putem utiliza firejail ca mai jos:

```
firejail --private firefox --private window
```

Firejail mai are și un panou de instrumente direct la nivel de desktop precum și un widget numit firetools (poate pentru începători în ale Linuxului).

Având în vedere că firejail practic intermediază relația între aplicație și sistemul de operare gazdă ea ne oferă o serie de comenzi suplimentare pentru inspecția și analiză rapidă a aplicațiilor care sunt gestionate de ea. De exemplu se poate verifica ce aplicații sunt executate sub controlul ei cu comanda

```
firejail --list
```

2209:bugs::/usr/bin/firejail /usr/bin/xfce4-notes
2549:bugs::firejail thunderbird

Această comandă verifică numai asocierile explicite nu și pe cele lansate manual de utilizator. De exemplu eu am shortcut-uri care lansează direct numai anumite aplicații cum ar fi firefox și care nu se observă în rezultatul execuției de mai sus.

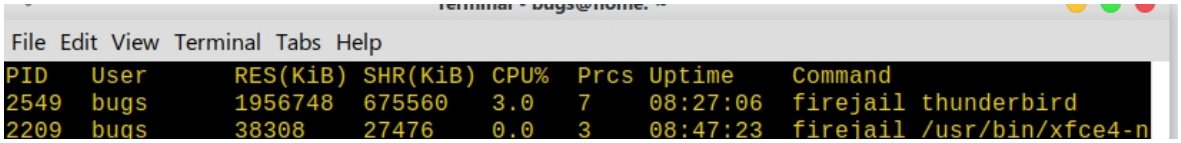
Pentru a vedea dacă comanda anterioară a mai scăpat ceva din vedere se poate folosi facilitatea de a inspecta arborele de procese gestionate de firefox.

firejail --tree
2209:xxx::/usr/bin/firejail /usr/bin/xfce4-notes
  2220:xxx::/usr/bin/firejail /usr/bin/xfce4-notes
    2256:xxx::/usr/bin/xfce4-notes
2549:xxx::firejail thunderbird
  2550:xxx::firejail thunderbird
    2665:xxx::/usr/lib/firefox-esr/firefox-esr
http://app.link.pentondes.com/e/er?s=1904481191&lid=94846&elqTrackId=2bbff9672ea64d76add7501740c7dd03&elq=3939ca0098314db1a4dc5bb853d68208&elqaid=26841&elqat=1
    ....

După cum se observă de această dată apar toate detaliile inclusiv cele legate de paginile deschise de copii.

În caz că se dorește o monitorizare mai detailată a resurselor utilizate de aplicațiile virtualizate (deoarece este posibil ca aplicațiile de task manager să măsoare încărcarea combinată dintre firejail si aplicație) se poate utiliza următoarea comandă:

firejail --top



Pentru a verifica gradul de încărcare a utilizării rețelelor se poate utiliza

firejail --netstats

Pentru a copia fișiere din și în interiorul sandbox-ului se poate utiliza –ls, –put și –get

Dacă se dorește impunerea de limitări ale traficului realizat de o aplicație aflată sub controlul firejail atunci se va da o comandă ca mai jos

firejail --bandwidth=mybrowser set eth0 80 20

care va limita viteza de download la 80KBps iar pe cea de upload la 20KBps

Firewall-ul Linux

Configurarea de firewall Linux care completează acest ansamblu de aplicații pentru asigurarea securității unui sistem a fost studiată deja în primul ciclu deci se prezumă a fi cunoscută. Pentru începători GUFW poate reprezenta o variantă mai comodă de configurare a firewall-ului.

Temă de laborator

Se va studia laboratorul și testa toate comenzile prezentate până acum. Apoi procesul va fi reluat cu alte aplicații la alegerea studentului.

Tema pe acasă

Fiecare student va trebui să prezinte cum a aplicat aceste cunoștințe pe sistemul lui inclusiv să explice maniera și regulile utilizate în configurarea firewall-ului sistemului personal. Cei care au sistem de operare principal MS - Windows oricum pentru această disciplină au nevoie de o mașină virtuală pe Linux și atunci vor prezenta aplicarea acestor tehnici la nivelul mașinii virtuale.