Inhoudsopgave

Lijst van Illustraties

Abstract

Implementatie van SELinux op een universele Linux server

door: Chris GEEBELEN
Pieter WUYTENS
promotor: ing. L. Rutten - docent

De opdracht voor ons eindwerk bestond erin een studie te maken van SELinux op Linux servers. SELinux is een uitbreiding op de standaard Linux beveiliging, ontwikkeld door de NSA (National Security Agency). Deze uitbreiding, die nu nog in een onderzoeksfase zit, is een zeer complexe en strikte manier van beveiligen, maar het ziet ernaar uit dat SELinux in de toekomst de traditionele beveiliging van Linux zal vervangen.
Eerst hebben we SELinux geïnstalleerd op Slackware, waarop we al onze testen hebben kunnen uitvoeren. Hierna hebben we zelf geprobeerd om bepaalde programma's te beveiligen.
Het schrijven van beveiligingsregels blijkt een zeer complexe zaak. Voor sommige programma's hebben we deze regels kunnen schrijven, voor andere programma's was dit veel te complex. Met beveiligingsregels kunnen restricties worden ingevoerd op wat men de programma's toelaat om uit te voeren, zoals het lezen van een directory of file.
Onderzoek naar beveiliging is een actueel thema dat in de toekomst zeker nog aan belang zal winnen. Onze conclusies uit dit eindwerk zijn:
- dat SELinux de standaardbeveiliging op Linux servers zal worden;
- dat SELinux zeer goed werkt: wat niet mag volgens de regels is ook echt niet mogelijk;
- dat SELinux vooralsnog zeer complex is en zeker nog gebruiksvriendelijker zal moeten worden.

Ref: E05/ELO/ 10

Dankwoord

Dit woord van dank is gericht aan iedereen die op één of andere manier heeft bijgedragen tot dit afstudeerwerk. Sommige mensen verdienen echter meer dan een gewoon dankwoordje.

Op de eerste plaats wensen wij onze interne promotor Leo Rutten te bedanken. Ondanks zijn drukke bezigheden heeft hij meermaals dit werk grondig doorgenomen, zijn opmerkingen en kritiek hebben bijgedragen tot de goede afloop van dit eindwerk. Met eventuele vragen konden wij steeds bij hem terecht.

Verder danken we alle mensen die met Linux en SELinux bezig zijn. Zonder hun toewijding zou Linux niet zo een goed besturingssysteem geworden zijn. Vooral willen wij Timothy Wood bedanken, hij is zowat de enige persoon die bezig is met SELinux op Slackware. Hij heeft ons hierbij zeer goed geholpen.

Onze vriendinnen Truus en Ingrid wensen we te bedanken voor het begrip, de steun en het geduld dat zij hebben opgebracht. Verder danken wij ook onze naaste familieleden voor de steun die wij van hen kregen tijdens de realisatie van dit werk.

Bedankt allemaal,

Pieter Wuytens en Chris Geebelen

Motivatie

Onze keuze om een eindwerk te maken over Linux servers heeft te maken met onze persoonlijke interesse. Er zijn twee redenen waarom we SELinux kozen. SELinux bezit enerzijds de laatste nieuwe ontwikkelingen inzake beveiliging van Linux servers. SELinux wordt anderzijds ontwikkeld door de programmeurs van de NSA/CSS[1], een organisatie waarvoor we veel belangstelling hebben.

Onze opdracht omvat niet alleen de implementatie van SELinux maar ook de effectieve installatie van een server. Vooraleer we SELinux op een werkende server installeren, moeten we het natuurlijk eerst installeren op een testserver. Ons eindwerk zal ons ongetwijfeld veel bijbrengen over servers. We moeten zoveel mogelijk diensten werkende krijgen, waarvan er zoveel mogelijk met SELinux beveiligd moeten zijn.

SELinux zal zeker de standaard worden voor de beveiliging van traditionele servers. Met ons eindwerk krijgen we dus een stapje voor op dit gebied, wat onze kans op een baan in de ICT branche kan vergroten.

We publiceren al onze informatie op het Internet om andere SELinux-gebruikers te helpen. Op deze manier dragen we bij tot een veiliger en stabieler Internet.

[1]NSA/CSS: The National Security Agency/Central Security Service, Amerikaanse staatsveiligheid. Voor meer uitleg zie 10.1

1 Inleiding

Om een computer te kunnen gebruiken, heb je een besturingssysteem (ook operating system of afgekort OS) nodig. Dit is het eerste programma dat gestart wordt als de computer wordt aangezet. Het is een samenhang van vele kleine programma's die ervoor zorgen dat andere programma's (applicaties) gestart en beëindigd kunnen worden. Het OS regelt ook de toegang tot de harde schijf, het scherm, de invoer van gegevens, ...
De computergebruiker gebruikt het besturingssysteem door middel van een commandoregel of door een grafische gebruikersinterface. Voorbeelden van besturingssystemen zijn: Windows, GNU[2]/Linux, Unix, MacOS. Voor ons project gebruiken wij GNU/Linux verder genoemd als Linux.

Linux bestaat uit een kernel die op zijn beurt bestaat uit code voor de basisfaciliteiten van Linux, bijvoorbeeld systeemaanroepen zoals `write()` en `read()`. Deze veelgebruikte basisfuncties zijn nodig om applicaties te kunnen schrijven zonder dat de programmeur zich hoeft te bekommeren om de details over hoe bijvoorbeeld een `read()`-actie (die invoer van een toetsenbord of een ander invoerapparaat leest) werkt. De kernel is gecompileerd en is dus een binair bestand.

Er bestaan verschillende soorten distributies van Linux. Voorbeelden van distributies zijn: Fedora, Slackware, Gentoo, … Het zijn allemaal besturingssystemen met een Linux kernel, maar er zijn verschillen tussen deze distributies.

Ons eindwerk handelt over SELinux. SELinux is de afkorting van Security Enhanced Linux. Het was in de eerste versies een uitbreiding op de bestaande Linux maar tegenwoordig is het een onderdeel van de kernel (vanaf kernel versie 2.6.1). SELinux zorgt voor een betere beveiliging van het systeem. Het werd ontwikkeld door het NSA/CSS. SELinux werkt door middel van mandatory access control (MAC) en role based acces control (RBAC). MAC wil zeggen dat regels bepalen wat gebruikers en programma's mogen op een systeem. Hieruit volgt dat er voor ieder nieuw programma regels gedefinieerd moeten worden. Dit zorgt ervoor dat het systeem op die manier heel goed beveiligd is tegen bijvoorbeeld: virussen, inbraakpogingen en

[2] GNU: de GNU licentie. Voor meer uitleg zie 10.2

andere ongerechtigheden. RBAC wil zeggen dat bepaalde users een rol bezitten. Met deze rol krijgt men toegang tot bepaalde domeinen, bijvoorbeeld: het administratordomein, het gebruikersdomein,…. Zo wordt de beveiliging bepaald naargelang de rollen die een gebruiker bezit.

Het geheel van regels en toelatingen wordt een policy genoemd. Deze policy bestaat uit verschillende logisch geordende bestanden.

SELinux had als eerste doel om servers te beveiligen. Een server is een computer die ten dienste van andere computers staat. Hierdoor kunnen er meerdere mensen tegelijk met een server verbonden zijn, daar waar dit niet mogelijk is op een gewone pc. Een server kan verschillende taken hebben. Voorbeelden van taken zijn: bestanden delen (fileserver), een printer delen (printserver), mails versturen en ontvangen (mailserver). Een server heeft meestal veel betere hardware dan een gewone pc voor thuisgebruik.

2 De opdracht

De opdracht van ons eindwerk omvat de volgende delen:

- Een studie over SELinux maken
- SELinux op verschillende distributies installeren
- SELinux op deze distributies testen
- Een programma schrijven en beveiligen
- Verklarende teksten in XML[3] schrijven

Het doel van ons eindwerk is een studie te maken over SELinux. We beschrijven hoe SELinux het systeem beveiligt, welke beveiligingsmodellen worden toegepast en hoe deze aangepast zijn voor gebruik in SELinux. Ook gaan we beschrijven waarom SELinux beter is dan de standaard Linux.

Het tweede deel van onze opdracht bestaat erin om SELinux op een aantal van de meest gebruikte distributies te installeren. Deze distributies zijn: Slackware, Fedora en Gentoo. We beschrijven welke bestanden men nodig heeft, welke configuratiebestanden men moet aanpassen en hoe men een nieuwe kernel moet compileren.

Vervolgens testen we SELinux op deze distributies. Hiertoe zullen labels van mappen en bestanden moeten worden aangepast; ook de policy en het aanmaken van nieuwe gebruikers vergen aanpassingen.

In het voorlaatste deel beschrijven we hoe we een klein programma, dat we zelf geschreven hebben, beveiligd hebben.

De laatste opdracht, en misschien wel de belangrijkste, is het schrijven van teksten. Ons doel is de teksten zo begrijpbaar en duidelijk mogelijk te maken zodat andere mensen die willen starten met SELinux een hulp aan onze teksten hebben. Deze teksten publiceren we op het Internet zodat iedereen over de hele wereld deze teksten kan raadplegen. Men kan onze teksten vinden op: http://ontwerpen1.khlim.be/~selinux.

[3] XML: Extensible Markup Language. Voor meer uitleg zie 10.3

3 De standaard Linux beveiliging

De standaard Linux beveiliging werkt met het DAC systeem. DAC staat voor Discretionary Access Control. Dit wil zeggen dat een gebruiker die eigenaar is van een object ook volledige controle heeft over dat object. Elk programma dat de gebruiker uitvoert, krijgt de rechten van de gebruiker. Bij dit model vertrouwt men de gebruiker. In Linux kan men dit terugvinden in de read/write/execute flags die toepasselijk zijn op user/group/others. Dit zijn flags die men kan veranderen met het commando `chmod` om zo de toegangsrechten te veranderen.

Er zijn een aantal veiligheidsproblemen bij DAC. Het eerste probleem is dat de gebruiker die eigenaar is van een object andere gebruikers toegang kan geven tot dat object. Nu moet men niet alleen de gebruiker, die eigenaar is, vertrouwen, maar ook alle andere gebruikers, en dat is niet veilig. Daarenboven moet men ook nog eens de programma's die de gebruikers uitvoeren vertrouwen want elk programma dat de gebruiker uitvoert heeft evenveel rechten als de gebruiker. Zo heeft elk programma bijvoorbeeld toegang tot het netwerk terwijl een programma dat niet noodzakelijk nodig heeft. Om een veilig systeem te verkrijgen kan men programma's dus beter alleen de toelatingen geven die het nodig heeft en de rest verbieden. Dit is niet mogelijk met DAC. Er zijn bijvoorbeeld programma's zoals `passwd` die tijdelijk root rechten krijgen (`setuid`, set user id programma) om de `passwd` file aan te passen. Hierbij krijgt dit programma dezelfde rechten als een root gebruiker en dus te veel rechten, wat natuurlijk niet veilig is. Men kan met dit systeem ook amper controleren wat mag en niet mag: men kan geen individuele toegang geven of programma's beperken in wat ze mogen. Nog een probleem bij dit systeem is dat de root user volledige toegang heeft tot alle objecten en alle instellingen kan wijzigen. Dus wanneer een hacker het wachtwoord van de root gebruiker heeft, heeft hij/zij volledige controle over het systeem.

4 De werking van SELinux

4.1 MAC

SELinux werkt met het mandatory acces control model, afgekort MAC. MAC zorgt ervoor dat de handhaving van de policy niet afhangt van een gewone gebruiker. Bij DAC was dit wel het geval omdat een gebruiker de rechten van files kan aanpassen. Het beveiligt informatie door security labels aan informatie te hangen en deze labels te vergelijken met het beveiligingsniveau van de gebruiker. Met MAC is het mogelijk programma's, processen, gebruikers en netwerk sockets te beveiligen. Bij DAC kan men alleen gebruikers beveiligen. MAC is veel veiliger dan DAC, maar DAC werkt sneller en is gemakkelijker te gebruiken.

4.2 Security context

Zoals hierboven is gezegd werkt SELinux met security labels. Een security label noemt men de security context van een object of subject. Een object is bijvoorbeeld een bestand, een socket of een IPC (interproces communication) object. Een subject is een proces. Een security context is informatie die gebonden is aan een file, proces of socket. Met deze security context en de policy kan SELinux beveiligingsbeslissingen nemen.

Een security context bestaat uit verschillende attributen. Hieronder wordt uitgelegd waaruit een security context bestaat.

4.2.1 Identity

Een identity onder SELinux is niet hetzelfde als de traditionele user id onder Linux. Ze kunnen samen bestaan onder hetzelfde systeem, maar zijn toch verschillend. Identities onder SELinux vormen een deel van de security context wat wil zeggen welke domeinen men kan binnengaan, wat men kan doen dus.
Men kan deze identity weergeven door het `id` commando uit te voeren onder SELinux.

Als men dit commando als een standaard gebruiker met naam chris uitvoert, een gebruiker met het minimum aan rechten, krijgt men dit:

```
Uid=1022(chris) gid=100(users) group=100(users)
context=chris:user_r:user_t
```

Chris is de identity in deze security context. Als gebruiker chris met het commando `su` (switch user) naar root overgaat dan ziet men dat de security context dezelfde gebleven is, alleen de Linux user id (uid) is veranderd:

```
Uid=0(root) gid=0(root) group=0(root)
context=chris:user_r:user_t
```

Wanneer de gebruiker chris toegelaten wordt in de `sysadm_r` rol en dan ook naar die rol overgaat met `newrole -r sysadm_r`, dan ziet met dit als men `id` uitvoert:

```
Uid=0(root) gid=0(root) group=0(root)
context=chris:sysadm_r:sysadm_t
```

Men ziet dus dat de security context wijzigt, maar de identity niet.

4.2.2 Domain

Elk proces wordt uitgevoerd in een domain. Een domain determineert welke toegang een proces heeft. Een domain is eigenlijk een lijst van wat processen mogen doen. Zo bestaat er een `sysadm_t` domain voor system administrators, een `user_t` domain voor gewone gebruikers, een `init_t` domain voor init, …

4.2.3 Type

Een type wordt toegekend aan een object en hiermee wordt bepaald wie toegang krijgt tot dat object. Een type is ongeveer hetzelfde als een domain, met het verschil

dat een domain toegepast wordt op een proces en een type op een object (file, directory, socket, enz.).

4.2.4 Rol

Een rol wordt toegekend aan een user. In de policy wordt bepaald welke user toegang heeft tot welke rol. Wanneer een bepaalde rol geen toegang heeft tot een bepaald domain dan wordt de toegang geweigerd.

4.2.5 Verschil tussen domain en type

Processen hebben een domain. Wanneer men de security context van een proces bekijkt is het laatste veld het domain. Als men bijvoorbeeld `passwd` uitvoert zal dat het `user_passwd_t` domain zijn.

Objecten zoals files, directories en sockets hebben een type. Dit kan men weergeven door `ls --context`. Het laatste veld is het type van dat object. Wanneer men een bestand aanmaakt in de home directory dan zal het type `user_home_t` zijn.

Hier kan verwarring ontstaan of het laatste veld in de security context een type of domain is. Neem nu de `/proc` directory. Alle processen hebben een domain, en `/proc` heeft een subdirectory voor elk proces. Elk proces heeft een security context. De security context van files en subdirectories van de `/proc` directory bevatten types, geen domains. Ook al stelt de `/proc` directory lopende processen voor, alles onder `/proc` zijn files en directories en hebben dan ook een type in plaats van een domain.

Wanneer men het `ls --context /proc` commando uitvoert krijgt men voor het init proces dit:

```
dr-xr-xr-x root root system_u:system_r:init_t 1
```

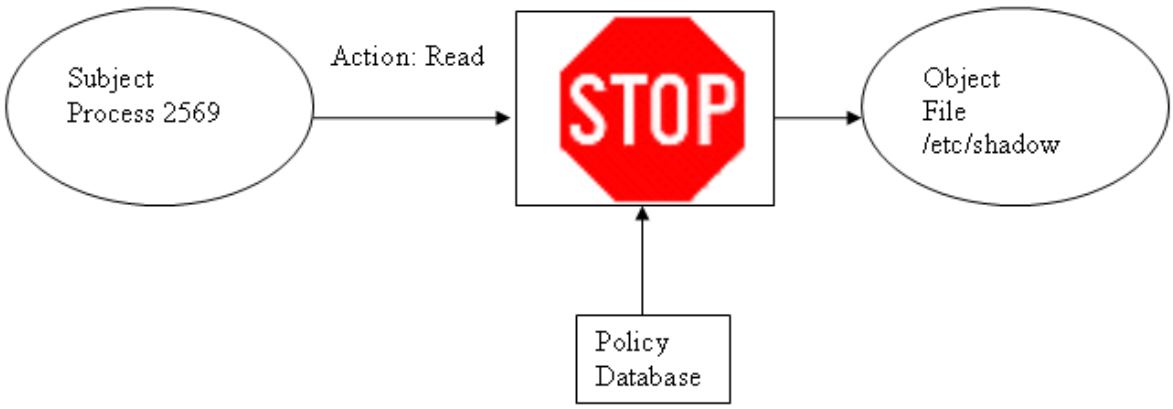
Het label of de security context geeft aan dat deze file het type `init_t` heeft. Maar het init proces wordt ook uitgevoerd in het `init_t` domain. Dus het type van het bestand is hetzelfde als het domain van het proces. En dit geld voor elk proces of bestand van `/proc`.

De commando's `chsid` (veranderen van security id) en `chcon` (veranderen van context) werken niet op de `/proc` directory omdat de `/proc` geen verandering van labels ondersteund.

Ook moeten we de daemons (processen die uitgevoerd worden op de achtergrond) starten via init. Het init proces is aangepast aan SELinux. Zo worden daemons in het juiste domain uitgevoerd. Dit is bij iedere distributie die wij getest hebben het geval. Hier zijn enkele voorbeelden van commando's om daemons te starten:

```
run_init /etc/init.d/ntpd start
run_init /etc/init.d/apache2 restart
run_init /etc/init.d/named stop
```

4.3 Hoe worden beslissingen genomen?



Figuur 1: De manier waarop beslissingen genomen worden

Zoals men in
Figuur 1 kan zien, zijn de beslissingen die moeten genomen worden van de aard: mag een bepaald proces een leesbewerking uitvoeren op een bepaald bestand? SELinux neemt deze beslissingen door middel van zijn policy.

Maar hierboven hebben we beschreven dat Linux ook een beveiligingssysteem heeft, namelijk DAC. DAC van Linux en MAC van SELinux werken samen. Wanneer er een beveiligingsbeslissing gemaakt moet worden zal eerst DAC geraadpleegd worden en daarna MAC. Mag de operatie van DAC niet, dan zal de operatie niet uitgevoerd worden. Mag de operatie van DAC wel, dan zal MAC geraadpleegd worden. Een operatie kan dus alleen maar uitgevoerd worden als ze zowel door DAC als door MAC toegelaten is.

4.4 Beveiligingsmodellen

SELinux werkt met een combinatie van beveiligingsmodellen. In dit deel worden deze modellen besproken.

4.4.1 TE model

Het TE model is de afkorting van Type Enforcement. Dit model zorgt voor een uitgebreide beveiliging van programma's. Het TE model geeft elk object een type en elk proces een domain. Het model beperkt de toegang van domains naar types en van domains naar domains. Beperkingen die het model kan opleggen zijn bijvoorbeeld het lezen en schrijven van bestanden en het lezen en aanmaken van directories. Deze toelatingen zet men in een aantal matrices. In de policy wordt ook bepaald in welke domains een gebruiker kan werken. Het TE model is het grootste deel van de policy. Types en domains worden bij dit model door elkaar gebruikt en worden als gelijkwaardige klassen beschouwd in SELinux.

Er bestaan een aantal TE regels die men in de policy gebruikt. De TE transistieregel zorgt voor toelatingen voor overgang van een domain naar een ander domain of van een type naar een ander type. Wanneer een gewone user een file aanmaakt in de

`/tmp` directory, dan zal deze file de security context `user_tmp_t` krijgen, hier heeft dus een overgang naar het `user_tmp_t` type plaatsgevonden.
Als een gewone user met het `user_t` domain ssh[4] uitvoert en daarna de security context weergeeft van ssh met het commando `ps ax --context` dan ziet men het volgende:

```
chris:user_r:user_ssh_t
```

Ssh wordt in het `user_ssh_t` domain uitgevoerd omdat het uitvoerbare bestand het type `ssh_exec_t` heeft en het `user_t` domain toegang heeft gekregen tot het `user_ssh_t` domain. Deze transitie wordt toegelaten door deze regel:

```
Role user_r types user_ssh_t;
```

De automatische transitie gebeurt met de volgende regel:

```
Domain_auto_trans(user_t, ssh_exec_t, user_ssh_t)
```

Al deze security contexts worden gegeven en gecontroleerd door middel van deze TE transistieregels.

Een ander soort regel is de TE acces vector regel. Met deze regel kan men toelatingen geven om bepaalde bestanden of directories te lezen of te schrijven.

Het TE model associeert gebruikers niet direct met domains. SELinux gebruikt het RBAC model om een extra laag te verkrijgen tussen gebruikers en domains. Het RBAC model wordt in het volgende deel besproken

[4] SSH: Secure Shell. Voor meer info zie 10.4

4.4.2 RBAC model

Een traditioneel RBAC model geeft gebruikers toegang tot bepaalde rollen en geeft elke rol bepaalde toelatingen. Zo kan men bijvoorbeeld een standaard user rol hebben: de `user_r` rol of een system administrator rol: de `sysadm_r` rol. Gebruikers die toegang hebben tot de system administrator rol hebben veel meer rechten dan een user die alleen toegang heeft tot de user rol. Het SELinux RBAC model geeft users toegang tot bepaalde rollen en geeft elke rol toegang tot bepaalde TE domains. Deze combinatie zorgt voor een vrij simpele controle over users (RBAC model) waarbij men gecompliceerde beveiligingen kan schrijven door middel van het TE model.

4.5 De SELinux policy

4.5.1 Wat is een policy

Een policy bestaat uit regels die bepalen welke rollen een user kan hebben, welke rollen welke domains kunnen binnengaan en welke domains welke types kunnen gebruiken. De huidige Slackware policy bestaat uit 16230 regels. In de policy wordt standaard niks toegelaten, als men iets wil toelaten moet men er regels voor schrijven. De policy wordt opgedeeld in een aantal files die tijdens het compileren worden samengebracht. Deze files worden in hoofdstuk 4.6 besproken. Het opdelen in files heeft als doel om dingen die samen horen samen in een file te zetten en het veel overzichtelijker te maken.

4.5.2 Wat kan men met een policy doen?

Door middel van een policy kan men een systeem configureren zoals men zelf wil. Men kan een gebruiker A toegang geven tot de `user_r` rol en de `sysadm_r` rol, en gebruiker B alleen toegang geven tot de `user_r` rol. Met een policy kan men controleren wat programma's kunnen doen en hoe programma's met elkaar reageren

door middel van toegang te geven tot files en dergelijke. Men kan dus bijvoorbeeld de policy zo configureren dat wanneer een bepaald domain een bestand in de `/tmp` directory schrijft, een ander domain het bestand niet kan lezen. Door middel van de policy geeft men programma's alleen maar die toelatingen die het nodig heeft om te werken. Zo maakt men het systeem zo veilig mogelijk. Wanneer een hacker de controle over een programma heeft zal hij/zij alleen maar die dingen kunnen doen die het programma mag.

4.5.3 Hoe een policy compileren?

Een policy moet gecompileerd worden naar een binair bestand vooraleer SELinux deze kan gebruiken. Om dit te doen moet men het `make` commando in de policy directory uitvoeren. Het compileerproces bestaat uit drie stappen.
Stap 1: Alle policy configuratie files worden aan elkaar gehangen. De policy configuratie files eindigen op .te en kan men vinden in de policy directory en subdirectories.
Stap 2: De m4 macro processor wordt op deze file losgelaten. Deze zal de `policy.conf` file aanmaken. De `policy.conf` file bestaat uit definities van types, domains en regels over wat domains kunnen doen en dergelijke.
Stap 3: In deze stap wordt de `policy.conf` gecompileerd met de checkpolicy policy compiler. Men kan de gecompileerde policy laten laden bij de volgende herstart met het commando `make install`, of men kan deze direct laten vervangen door het commando `make load` of `make reload`. Men kan de policy ook alleen laten compileren met het `make policy` commando. Deze commando's werken alleen maar als men ze in de policy directory uitvoert.

4.6 Policy files

Hieronder worden de belangrijkste files van de policy uitgelegd. Deze bestanden kan men bij Slackware en Gentoo in de directory `/etc/security/selinux/src/policy`, of een van de subdirectories vinden. Bij Fedora staan de policy files van de strict policy in de directory `/etc/selinux/strict/src/policy`.

En voor de targeted policy in de `/etc/selinux/targeted/src/policy`.

4.6.1 Attributen

Attributen worden in de `attrib.te` file gedeclareerd. In deze file vindt men het `domain` attribuut. Dit attribuut staat voor elk type dat men aan een proces kan toewijzen.

Het `privuser` attribuut laat een domain toe om van SELinux user identity te veranderen. De volgende domains kunnen van SELinux user identity veranderen: `sysadm_su_t, initrc_su_t, staff_su_t, run_init_t, local_login_t, remote_login_t, sshd_t, sshd_extern_t` en `xdm_t`.

```
Constrain proces transition { u1==u2 or t1==privuser };
```

In de bovenstaande regel kan men zien dat de user identity steeds dezelfde blijft bij een proces transitie, tenzij het domain het attribuut privuser bezit.

Met het `privrole` attribuut kan een domain van SELinux rol veranderen. Newrole bijvoorbeeld heeft het `privrole` attribuut nodig om van rol te veranderen. Met dit attribuut kan men alleen naar andere user rollen veranderen. Dit wordt bepaald met met de onderstaande regel:

```
Constrain proces transition { r1==r2 or t1=privrole };
```

Het `priv_system_role` attribuut laat toe om naar de `system_r rol` over te gaan.
Het `privowner` attribuut zorgt ervoor dat een domain een andere SELinux user identity aan een file kan geven of een file kan creëren met een identity die niet hetzelfde is als de proces identity. Het `passwd_t` process heeft de identity van de user die het uitvoert, maar het wil de `/etc/shadow` file herlabelen met de `system_u` identity en heeft daarvoor het `privowner` attribuut nodig.

4.6.2 User files

In dit deel gaan we de bestanden beschrijven die betrekking hebben tot users.

4.6.2.1 De users file

De users file bevat definities van users die herkend worden door het SELinux systeem. Wanneer een user in deze file gedeclareerd is zal de naam van de user in de security context komen. Wanneer dat niet zo is zal de user de `user_u` identity krijgen. In deze file vindt men de volgende regel:

```
User root roles { staff_r sysadm_r };
```

Dit wil zeggen dat user root in de `staff_r` en `sysadm_r` rol kan werken. Veranderen van rol kan men met het newrole programma doen.

```
User Chris roles { staff_r sysadm_r };
```

De bovenstaande regel zorgt ervoor dat user Chris de `sysadm_r` en `staff_r` rol kan hebben. Dit voorbeeld toont aan dat een root user niet noodzakelijk de `sysadm_r` rol heeft omdat hij root is. Wanneer men de root user niet toelaat in de `sysadm_r` rol dan zou user Chris meer macht hebben dan de root user.

In deze file kan men ook nog de volgende regel vinden:

```
User system_u roles system_r
```

De `system_u` identity is een identity voor processen en objecten. Men mag nooit de `system_u` identity aan een user proces geven omdat de `system_u` voor daemons is. Wanneer een gebruiker de `system_u` identity heeft, heeft hij toegang tot de `system_r` rol en eender welk daemon domain. Dit is natuurlijk niet veilig. Er is ook nog een regel die betrekking heeft op alle users die wel in `/etc/passwd` staan maar niet in deze `users` file. Users die niet in deze file zitten zullen automitisch deze role toegewezen krijgen.

```
User user_u roles { user_r } ;
```

4.6.2.2 De user.te file

Deze file kan men vinden in de subdirectory `domains` van de policy directory. Ze bevat de domains met weinig rechten. In deze file vindt men deze regel:

```
Full_user_role(user)
```

Deze regel laat alle noodzakelijke dingen toe om de user rol standaard dingen te laten doen zoals `bin_t` programma's uitvoeren in hun home directory. De `full_user_role` macro wordt uitgelegd in 4.6.2.3.

```
full_user_role(staff)
allow staff_t unpriv_userdomain:process signal_perms;
can_ps(staff_t, unpriv_userdomain)
allow staff_t { ttyfile ptyfile tty_device_t }:chr_file
getattr;
```

De bovenstaande regels definiëren het `staff_t` domain. De tweede regel laat het `staff_t` domain toe om signalen te zenden naar processen die in domains met weinig rechten draaien zoals het `user_t` en het `staff_t` domain. Het `staff_t` domain kan `ps` uitvoeren en alles zien in het `user_t` en andere user domains als die er zijn, waar `user_t` dat niet kan. De laatste regel laat `staff_t` toe toegang te hebben tot de attributen van elk terminal device.

```
dontaudit unpriv_userdomain sysadm_home_dir_t:dir { getattr
search };
```

Deze regel zorgt ervoor dat wanneer het user domain toegang probeert te hebben tot de `/root` directory bijvoorbeeld door middel van `ls -l, cd /root` of toegang tot een file onder de `/root` directory, dit niet gelogd word.

Meestal zal een systeem administrator deze files niet moeten aanpassen

4.6.2.3 De user_macros.te file

Deze file bevat macro's voor user login domains. Dit zijn domains die users krijgen na het inloggen. Men kan deze file in de subdirectory `macros` vinden.

We delen deze file op in stukken en verklaren elk stuk. Het eerste deel dat we gaan verklaren gaat over macro's voor user login domains.

```
# user_domain() is also called by the admin_domain() macro
undefine(‘user_domain’)
define(‘user_domain’,‘
```

De bovenstaande regels definiëren de macro `user_domain`. Deze macro definieert een domain voor niet administratieve gebruikers.

```
# Use capabilities
ifdef(`single_userdomain', `
# if we have a single user domain then gpg needs SETUID
# access.  Also lots of
# other things will have similar issues.
allow $1_t self:capability setuid;
')dnl end single_userdomain
```

Capability wil in de eerste regel zeggen: de mogelijkheid om de user id te veranderen (`setuid`). Een programma kan `setuid()` oproepen om zijn user id te veranderen, maar dan moet het wel eerst de mogelijkheid hebben om dat te doen, of deze mogelijkheid niet opgegeven hebben. Een programma kan `setuid()` niet oproepen als deze mogelijkheid niet toegelaten is.

In het tweede deel worden macro's beschreven voor gewone user domains

```
undefine(`full_user_role')
define(`full_user_role', `
```

```
# user_t/$1_t is an unprivileged users domain.
type $1_t, domain, userdomain, unpriv_userdomain,
web_client_domain;
```

Met de eerste 2 regels wordt de macro `full_user_role` gedefinieerd. Men definieerd het type $t_t en geeft het vier attributen.

```
# Read /etc.
allow $1_t etc_t:dir r_dir_perms;
allow $1_t etc_t:notdevfile_class_set r_file_perms;
allow $1_t etc_runtime_t:{ file lnk_file } r_file_perms;
```

Hier geeft men de `user_t` toegang om `/etc` te lezen. Men kan de files lezen en dingen doen zoals `ls -l` in de `/etc` directory. Met de `notdevfile_class_set` kan men niet device files lezen zoals files, symlinks, socket files en fifo files. `Etc_runtime_t` is het type voor sommige files in de `/etc` directory.

```
undefine(`in_user_role')
define(`in_user_role', `
role user_r types $1;
role staff_r types $1;
')
```

Definieert de macro `in_user_role`. De macro moet gebruikt worden in de `.te` file van het domain. Als men de `passwd.te` file bekijkt, ziet men dat de `in_user_role` macro is gebruikt met de parameter `passwd_t`. Dit is hetzelfde als:

```
Role user_r types passwd_t;
Role staff_r types passwd_t;
```

Dit wil zeggen dat de `user_r` en de `staff_r` het `passwd` programma kunnen uitvoeren.

4.6.3 System administrator files

In dit deel beschrijven we de policy files die betrekking hebben tot de `sysadm_r` rol, de system administrator rol.

4.6.3.1 De admin_macros.te file

```
undefine(`admin_domain')
define(`admin_domain',`
```

Deze twee regels definieren de `admin_domain` macro. Deze macro definieert een domain voor een administratieve gebruiker.

```
allow $1_t policy_config_t:dir { getattr search };
allow $1_t policy_config_t:file { getattr unlink };
```

Hiermee kan `sysadm_t` getattr doen (dingen zoals `ls -l`) en files en directories zoeken onder een directory met type `policy_config_t`.

```
allow $1_t kernel_t:system syslog_read;
```

Laat `sysadm_t` toe om systeem logs te lezen. `Kernel_t` is het type van de kernel, system is de class van de operatie om de syslog te lezen.

```
# Use capabilities other than sys_module.
allow $1_t self:capability ~sys_module;
```

Deze regels laten `sysadm_t` toe om alle mogelijkheden te gebruiken behalve die van `sys_module`, welke gebruikt worden om modules te laden.

```
# Get security policy decisions.
can_getsecurity($1_t)
```

Als we nu `can_getsecurity` opzoeken in de file `core_macros.te` dan zien we er dit:

```
# can_getsecurity(domain)
```

```
#
# Authorize a domain to get security policy decisions.
#
define(`can_getsecurity',`
allow $1 security_t:dir { read search getattr };
allow $1 security_t:file { getattr read write };
allow $1 security_t:security { check_context compute_av
compute_create compute_relabel compute_user };
')
```

Hier geeft men $1 de toelating om alle directories van type `security_t` te lezen, attributen op te vragen en te zoeken. $1 kan ook attributen opvragen, lezen en schrijven van files van het type `security_t`. Als laatste kan $1 de geldigheid van een context controleren, controleren of de policy de source context toegang geeft tot de target context, berekenen van een context voor het labelen van een nieuw object, berekenen van de nieuwe context wanneer men een object relabeld, en determineren of een user context bereikt kan worden met een bepaalde source context.

```
# Change system parameters.
can_sysctl($1_t)
```

Het `sysadm_t` domain kan `sysctl` parameters veranderen, dat is ongeveer alles onder `/proc/sys`. De `can_sysctl` macro kan men vinden in de `global_macros.te` file.

4.6.4 De file_contexts directory

Deze directory bevat nog 2 subdirectories: `misc` en `program`.

De `program` subdirectory bevat bestanden die security contexts beschrijven van files die deel zijn van geïnstalleerde packages of programma's.

De `misc` subdirectory bevat nog een aantal specificaties. Bij sommige distributies is deze directory zelfs leeg.

In de `file_contexts` directory vindt men nog 2 bestanden: `file_contexts` en `types_fc`. De `file_contexts` file wordt automatisch gecreëerd nadat de policy gecompileerd is.

De `types.fc` file bevat security contexts voor systeem files en user home directories. Wanneer files gelabeld worden zal het pad van de file vergeleken worden met de regels in deze file. Wanneer er een overeenkomstige regel is gevonden, zal de file de context krijgen zoals die in deze file beschreven is, anders zal deze file niet herlabeld worden. Als er meerdere overeenkomstige regels gevonden zijn, dan zal de laatste regel toegepast worden. Natuurlijk worden alle files gelabeld. De files die geen context gespecificeerd hebben krijgen de `system_u:object_r:default_t` context. Hieronder volgt een voorbeeld:

```
/               -d   system_u:object_r:root_t
```

Deze regel zorgt ervoor dat de root directory gelabeld wordt met de `system_u:object_r:root_t` context. In de middelste kollom kunnen de volgende tekens voorkomen:

-- alleen voor gewone bestanden
-b alleen voor block device bestanden
-c alleen voor character device bestanden
-d alleen voor directories
-s alleen voor socket bestanden

Wanneer er geen teken voorkomt in de middelste kollom dan wil dat zeggen dat alle files en directories deze context krijgen.

4.6.5 De types directory

In de `types` directory worden types gedefinieerd die niet geassocieerd zijn met een bepaald domain. De `types` directory bevat de volgende files:

4.6.5.1 Device.te

De `device.te` file definieert types met betrekking tot apparaten en de files met betrekking tot apparaten (devices).

4.6.5.2 Devpts.te

In de `devpts.te` file worden types beschreven die betrekking hebben tot het `/dev/pts` filesystem.

4.6.6 De macros directory

De `macros` directory bevat m4 macro's die in TE files worden gebruikt. De `user_macros.te` en `admin_macros.te` files hebben we al besproken. Nu gaan we de twee andere files in die directory bespreken.

4.6.6.1 Core_macros.te

In de `core_macros.te` file vindt men macro's die men normaal niet veranderd en het is ook aangeraden dat niet te doen. Dit doet men omdat de kern van de policy in deze file staat en als men policy's wil uitwisselen zou deze kern hetzelfde moeten zijn.

Sommige van de macro's bevatten macro's voor het groeperen van classen en toelatingen:

```
define(`dir_file_class_set', `{ dir file lnk_file sock_file
fifo_file chr_file blk_file }')
```

Deze regel definieerd de macro `dir_file_class_set` en bevat classen voor `dir` (voor directories), `file` (voor files), `lnk_file` (voor symbolische links), `sock_file` (voor unix domain sockets), `fifo_file` (voor named pipes), `chr_file` (voor character devices) en `blk_file` (voor block devices).

```
define(`rw_file_perms', `{ ioctl read getattr lock write
append }')
```

De macro `rw_file_perms` bevat toelatingen voor `ioctl`, `read` (file lezen), `getattr` (attributen opvragen), `lock`, `write` en `append`.

4.6.6.2 Global_macros.te

In de `global_macros.te` file vindt men macro's die toepassing hebben op het volledige systeem.

```
# can_setexec(domain)
define(`can_setexec',`
allow $1 self:process setexec;
allow $1 proc_t:dir search;
allow $1 proc_t:{ file lnk_file } read;
allow $1 self:dir search;
allow $1 self:file { read write };
')
```

Dit is de macro `can_setexec`. $1 kan de execute context veranderen, dus het kan de context van een kindproces bepalen. $1 kan ook de `/proc` doorzoeken en kan files en symbolische links lezen in deze directory.

4.6.6.3 De /macros/program directory

De `/macros/program` directory bevat aanvullende macro's voor programma's die een policy per gebruiker nodig hebben. Programma's zoals ssh hebben een policy per gebruiker nodig omdat het afgeleide domain afhangt van het opgeroepen gebruikersdomain. Wanneer men de `ssh_macros.te` file bekijkt ziet men dit:

```
define(`ssh_domain',`
type $1_ssh_t, domain, privlog;
```

Wanneer het oproepende domain `user_t` is, dan zal het afgeleide domain `user_ssh_t` zijn. Hetzelfde voor wanneer `staff_t` het oproepende domain is, dan zal het afgeleide domain `staff_ssh_t` zijn.

```
domain_auto_trans($1_t, ssh_exec_t, $1_ssh_t)
```

Deze regel laat ssh toe om van het oproepende domain over te gaan naar het afgeleide domain.

4.6.7 De flask directory

Deze directory bevat 3 belangrijke files:

- `initial_sids`
- `security_classes`
- `access_vectors`

Hieronder bespreken we deze files.

De `initial_sids` file specificeert een aantal initial SID waardes. Deze waardes worden gebruikt om objecten te labelen tijdens het opstarten van het systeem. Systeem administrators moeten deze file normaal niet aanpassen.

De `security_classes` file definieert een aantal object classen zoals `file` en `dir`, dit zijn de meest gebruikte classes. Ook deze file zal men meestal niet moeten aanpassen.

De laatste file in deze directory is de `acces_vector` file. Deze file specificeert een 150 tal verschillende operaties die men kan uitvoeren op de ongeveer 30 gedefinieerde classes. De meest gebruikte operaties zijn `read`, het lezen van een file of ander object en `write`, het schrijven van een file of een ander object.

4.7 SELinux praktisch

Al deze testen hebben we in Slackware 9.1 gedaan, natuurlijk met SELinux geïnstalleerd zoals in hoofdstuk 5.2 uitgelegd is.

4.7.1 Inloggen

Zoals eerder gezien wordt in de `users` file gespecificeerd welke user toegang heeft tot welke rol. De root user kan volgens die file de rollen `sysadm_r` en `staff_r` aannemen. Wanneer men inlogt, krijgt men de vraag als men een andere rol wil kiezen of gewoon de standaard `sysadm_r` rol wil aannemen. Men kan kiezen uit de volgende rollen: `sysadm_r` en `staff_r`. Wanneer men zich inlogt met de `sysadm_r` rol en daarna `id` uitvoert dan krijgt men dit te zien:

```
Root:sysadm_r:sysadm_t
```

Als nu een gewone gebruiker Chris zich inlogt, kan deze niet kiezen uit verschillende rollen. Hij komt standaard in de `user_r` rol.

De standaard SELinux policy heeft 4 rollen:

`Staff_r`: Wordt gebruikt voor gebruikers die toegelaten worden in de `sysadm_r` rol.

`Sysadm_r`: Wordt gebruikt voor systeem administrators.

`System_r`: Wordt gebruikt voor systeemprocessen en objecten.

`User_r`: Wordt gebruikt voor gewone gebruikers.

Natuurlijk kan men in de policy nog extra rollen definiëren, maar dat wordt meestal niet gedaan.

4.7.2 Context veranderen

Men kan de context al kiezen bij het inloggen, maar men kan deze na het inloggen ook nog met het `newrole –r` commando wijzigen.
Als root van context wil veranderen kan hij dat met dit commando:

```
Newrole –r staff_r
```

Wanneer hij dit ook doet en dan `id` uitvoert dan ziet men het volgende:

```
Context=root:staff_r:staff_t
```

Bij het veranderen van rol wordt altijd het wachtwoord van de gebruiker gevraagd.
Wanneer gebruiker Chris het volgende commando uitvoert:

```
Newrole –r sysadm_r
```

zal SELinux een foutmelding geven omdat deze gebruiker geen toegang heeft tot die rol. Na het veranderen van rol kan men de context altijd controleren met het `id` commando.

De policy laat niet toe om iemand met de `user_r` rol naar de `sysadm_r` rol over te laten gaan.

4.7.3 Security contexts opvragen

SELinux heeft een aantal gewone commando's aangepast om de security context weer te geven. Deze commando's zijn:

`Id`: de user security context bekijken.

`Ls`: de file security context bekijken.

`Ps`: de proces security context bekijken.

Met het `id` commando krijgt men de volledige security context van een user met user id en group id. Het `id` commando heeft een nieuwe optie, de `id –Z` optie. Met deze optie krijgt men alleen de security context te zien.

Het `ls` commando geeft een lijst van de inhoud van een directory. Dit commando heeft de volgende opties:

`--context`: geeft een deel van de file context weer zodat deze op een lijn past.

`--lcontext`: geeft de volledige file context weer.

`--scontext`: geeft alleen de security context van de file weer.

`-Z`: zelfde resultaat als `--context`.

Het `ps` commando heeft twee nieuwe opties: `-Z` en `–context`. Deze opties doen allebei hetzelfde: ze geven de security context van processen weer.

4.7.4 Permissive en enforcing mode

Permissive mode is een mode waarbij SELinux de overtredingen alleen maar logt in een file maar niet tegenhoud. Deze overtredingen worden gelogd in de `/var/log/syslog` file. De inhoud van deze boodschappen zullen we in 4.7.8 bespreken. De permissive mode wordt gebruikt om policy's te debuggen.

Bij enforcing mode wordt de policy ook echt toegepast. Wat niet mag volgens de policy wordt tegengehouden en gelogd. Men kan zelfs een kernel compileren zonder `CONFIG_SECURITY_SELINUX_DEVELOP` support, wat wil zeggen dat men SELinux niet in permissive mode kan opstarten. Dit mag men alleen doen als men zeker is dat de policy goed werkt.

Men kan van permissive naar enforcing mode gaan met het commando:

```
Echo 1 > /selinux/enforce
```

Of met het commando:

```
Setenforce 1
```

Wanneer men de 1 door een 0 vervangt kan men terug van enforcing naar permissive gaan. Om te zien in welke mode SELinux werkt kan men dit commando uitvoeren:

```
Cat /selinux/enforce
```

Natuurlijk moet men in de `sysadm_r` rol zitten om deze commando's uit te voeren.

Men kan definiëren in welke mode SELinux opstart. Dit doet men door in het boot menu de volgende parameter mee te geven: `enforcing`. Wanneer men deze parameter 1 maakt dan zal SELinux opstarten in enforcing, anders in permissive. Als men bijvoorbeeld `lilo` gebruikt dan kan men deze regel toevoegen:

```
Append="enforcing=0"
```

Nu zal SELinux in permissive mode opstarten.

4.7.5 Vergelijking van commando's uitgevoerd in verschillende rollen

Wanneer iemand met de `user_r` rol in enforcing mode `ps ax --context` uitvoert krijgt hij alleen maar zijn processen te zien die in de domains werken waar de `user_r` rol toegang tot heeft. Zoals eerder al gezegd heeft ieder proces een directory in de `/proc`. Hij kan dus alleen die processen zien waar hij toegang toe heeft in de `/proc`.

Als iemand in het `sysadm_t` domain `ps ax --context` uitvoert zal hij alle processen zien waar die gebruiker toegang toe heeft, dus ook processen van het `user_t` domain. Dus iemand in het `user_t` domain kan geen processen zien van het `sysadm_t` domain. Dit is een voordeel. Want als `user_t` alle processen kon zien die in uitvoering zijn, dan zou hij een proces kunnen zien waar mogelijke

beveiligingsfouten in zitten. Maar nu dat een `user_t` deze niet kan zien is het risico dat hij deze beveiligingsfout gaat gebruiken al veel kleiner geworden.

4.7.6 Nieuwe user account aanmaken

We gaan in dit deel beschrijven hoe we een nieuwe user met naam Pieter gaan aanmaken. Een gebruiker in het `sysadm_t` moet het volgende commando uitvoeren:

```
adduser –d /home/pieter/ -g users –s /bin/bash pieter
```

`-d` voor de home directory te kiezen
`-g` voor de user ook aan een groep toe te wijzen
`-s` voor een shell te kiezen hier bash

Daarna moet men

```
Passwd pieter
```

uitvoeren om de user een paswoord te geven.

In de volgende stap gaan we pieter de `user_r` rol toewijzen. We openen de `/etc/security/selinux/src/policy/users` file met het commando:

```
Pico /etc/security/selinux/src/policy/users
```

Op het einde van dit bestand moet men deze regel bijvoegen:

```
User pieter roles { user_r };
```

Wanneer men meerdere rollen wil toewijzen aan deze user dan kan men die tussen de haken bijvoegen. Bijvoorbeeld:

```
User pieter roles { user_r sysadm_r };
```

Nu kan pieter ook in de sysadm_r rol komen.
We hebben de policy aangepast en moeten de nieuwe policy dus laden met het commando:

```
Make load
```

In de volgende stap gaan we pieter een standaard security context geven.

We openen de /etc/security/default_contexts file. In deze file zien we deze regel staan:

```
System_r:local_login_t   user_r:user_t
```

Wanneer een gebruiker zich inlogt, zal het /bin/login programma dat in het local_login_t domain draait de gebruiker automatisch het user_t domain geven.
Deze regel veranderen we in:

```
System_r:local_login_t sysadm_r:sysadm_t user_r:user_t
```

Als de user toegang heeft tot het sysadm_t domain dan zal het /bin/bash programma automatisch het sysadm_t domain toewijzen aan die user.

In deze file vindt men ook nog een lijn:

```
System_r:sshd_t     user_r:user_t staff_r:staff_t
```

Dit wil zeggen dat een gebruiker die zich inlogt via ssh standaard in het user_r domain komt.

In deze laatste stap gaan we de home directory van de user herlabelen. Standaard heeft de /home/pieter directory het domain default_t. Men moet in de /etc/security/selinux/src/policy directory make relabel uitvoeren of het volgende commando uitvoeren:

```
Setfiles
/etc/security/selinux/src/policy/file_contexts/file_contexts
/home/pieter
```

Daarna heeft de map van pieter de volgende context:

```
Pieter:object_r:user_home_dir_t
```

Dit is de juiste context.

4.7.7 Filesystems en files labelen

Alle files worden normaal tijdens het installeren van SELinux gelabeld. Maar soms is het nodig om filesystems en files te herlabelen na de installatie. Bijvoorbeeld tijdens de installatie van een nieuw filesystem moet dit filesystem gelabeld worden. Of wanneer men met een niet-SELinux kernel geboot heeft zijn er files aangemaakt zonder security context. Men kan deze files labelen met het `make relabel` commando of een van de hieronder beschreven commando's. Het `make relabel` commando neemt veel tijd in beslag afhankelijk van het aantal files op de harde schijf. Als men dus maar een paar files wil herlabelen of labelen kan men een van deze commando's gebruiken:

`/usr/bin/chcon:`	Labelt een of meerdere files met een gespecificeerde security context
`/sbin/fixfiles:`	Labelt alle filesystems volgens de `src/policy/file_contexts/file_contexts` file.
`/sbin/restorecon:`	Labelt een of meerdere files volgens de `src/policy/file_contexts/file_contexts` file.
`/usr/sbin/setfiles`:	Labelt een of meerdere files of filesystems volgens de inhoud van een bepaalde file.

Het **chcon** programma labelt een of meerdere files met een gespecificeerde security context. Het commando bestaat in twee vormen. De eerste vorm is het labelen van een file met een gespecificeerde security context en de tweede vorm is het labelen

van een file met de security context van een referentiefile. Wanneer men de `/etc/hosts` file een bepaalde security context wil geven kan dit met:

```
Chcon system_u:object_r:etc_t /etc/hosts
```

Of men kan de `/etc/hosts.allow` file labelen met dezelfde context als de `/etc/hosts` file.

```
Chcon –reference=/etc/hosts /etc/hosts.allow
```

Het **fixfiles** programma labelt alle beschikbare filesystems volgens de inhoud van de standaard file: `src/policy/file_contexts/file_contexts`. Het `fixfiles` commando heeft de volgende opties:

`Check:` Geeft de incorrecte file labels weer maar verandert ze niet.
`Restore:` Past de incorrecte labels aan.
`Relabel:` Herlabeld alle beschikbare filesystems

Het **restorecon** programma labelt een of meerdere files volgens de `src/policy/file_contexts/file_contexts` file. Als men bijvoorbeeld de `/etc/hosts` file wil labelen met het standaard label doet men dit met het volgende commando:

```
Restorecon /etc/hosts
```

Het fixfiles programma labelt alle beschikbare filesystems, maar het **setfiles** programma labelt een of meerdere filesystems.

```
Setfiles src/policy/file_contexts/file_contexts /etc/hosts
```

4.7.8 De SELinux foutmeldingen

In dit deel gaan we de meldingen die SELinux in de `/var/log/syslog` file geeft, uitleggen. Als een bepaalde operatie niet toegelaten is door SELinux zal SELinux

een melding in deze file geven. Men kan deze meldingen ook laten weergeven door het `dmesg` commando. Nog een snellere methode is om `/etc/syslog.conf` aan te passen zodat we alle errors op een virtuele console zien. We gaan de meldingen uitleggen aan de hand van enkele voorbeelden.

Voorbeeld 1

```
Avc: denied {write} for pid=10400 exe=/usr/bin/nmap lport=255
scontext=root:staff_r:nmap_t tcontext=root:staff_r:nmap_t
tclass:rawip_socket
```

Deze boodschap wil zeggen dat een `write` operatie niet toegelaten was. Het proces dat deze operatie wou uitvoeren was `nmap`. De security context van het proces is `root:staff_r:nmap_t` en de security context van het object is `root:staff_r:nmap_t`. De class van het object is `rawip_socket`. Nog andere classen zijn: file, dir, …
We kunnen deze fout oplossen door deze regel bij `nmap.te` toe te voegen.

```
Allow nmap_t self:rawip_socket { write };
```

Voorbeeld 2

```
avc: denied { getattr } for pid=6011 exe=/usr/bin/vim
path=/etc/shadow dev=03:03 ino=123456 \
scontext=chris:user_r:user_t
tcontext=system_u:object_r:shadow_t tclass=file
```

Deze melding geeft weer dat het opvragen van attributen niet toegelaten was. Het programma dat deze attributen wou opvragen is `vim`. De file waarvan `vim` de attributen wou opvragen is `/etc/shadow`. De context van `vim` is `Chris:user_r:user_t` en de context van `/etc/shadow` is `system_u:object_r:shadow_t`. Dit kunnen we ook weer oplossen met een allow regel. Normaal wordt dit niet gedaan omdat het niet de bedoeling is dat een gebruiker `/etc/shadow` kan lezen.

```
Allow user_t shadow_t:file { getattr };
```

Nu zal er een volgende error komen die zal zeggen dat een user de file niet mag lezen. Dus deze file is echt wel goed beveiligd. We kunnen de user toelaten deze file te lezen door naast `getattr` nog `read` te zetten.

In deze log file kunnen ook meldingen komen van operaties die geslaagd zijn. Dit kan men in de policy specificeren met de regel `auditallow`. Als men bijvoorbeeld overschakelt van permissive naar enforcing zal daar een melding van gemaakt worden.

5 Installatie van SELinux op verschillende distributies

5.1 Fedora Core 3

5.1.1 Maken van partities

Het installeren van Fedora is niet zo moeilijk. Het is vergelijkbaar met het installeren van Windows. Op een gegeven moment krijg je ook de kans om SELinux mee aan te zetten. Bij andere distributies komt hier wel iets meer bij kijken. Hier volgt een korte beschrijving voor de installatie van SELinux op Fedora.

Het eerste wat we moeten doen is natuurlijk de juiste partities maken voor SELinux. Dit kan je met verschillende programma's doen. In Windows kan je dit doen met partition magic. Wij hebben het gedaan met fdisk onder Linux.

Dit zijn de stappen die we met fdisk hebben gezet:

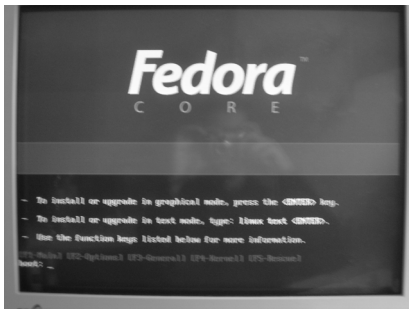
```
fdisk /dev/hda
```

Dev staat voor device, hd voor hard disk en a staat voor de eerste harde schijf.
Nu zitten we in het "menu" van fdisk.
Met "p" (print) zien we welke partities er al zijn en welke we nog willen bijmaken. Als er geen ruimte meer vrij is op de harde schijf dan kan je partities verkleinen met partition magic ofwel gewoon de partities allemaal verwijderen en opnieuw maken. Wij hadden natuurlijk ruimte voor Linux voorzien.
Met "n" (new) kan je een nieuwe partitie aanmaken. Dan typen we het nummer van de partitie in: in ons geval is dat 4. Daarna moeten we de grootte opgeven. We drukken eerst gewoon op enter zodat fdisk standaard het eerste vrije blok gaat gebruiken. Daarna typen we +1024M, dan reserveert fdisk een ruimte van 1024 MB of 1 GB.
Met "t" (type) veranderen we het type van de partitie.

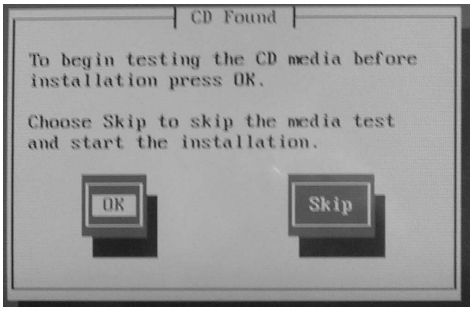
Met "l" (list) krijgen we de lijst van de mogelijke types te zien . We maken deze partitie van het type swap (82).
Met "n" (new) gaan we weer een nieuwe partitie maken om onze Linux op te installeren. Deze partitie komt op plaats 3. Vervolgens drukken we twee maal op enter (gebruikt de rest van de beschikbare ruimte). Het type staat standaard goed op 83.
Met "a" (active) kunnen we deze partitie actief maken. We moeten wel de juiste partitie actief maken nl. partitie 3.
Met "w" (write) kunnen we de partitie tabel opslaan.
Nu kunnen we beginnen met het installeren van Linux. We moeten deze procedure eigenlijk iedere keer doen, maar omdat we nu onze partities hebben gemaakt kunnen we deze voor de volgende installaties ook gebruiken.

5.1.2 Starten van de installatie

We starten de installatie door gewoon de eerste cd van Fedora Core 3 in de cd-rom drive te steken en de computer opnieuw op te starten. Dit is het eerste scherm dat we zien.



Figuur 2: Fedora opstartscherm

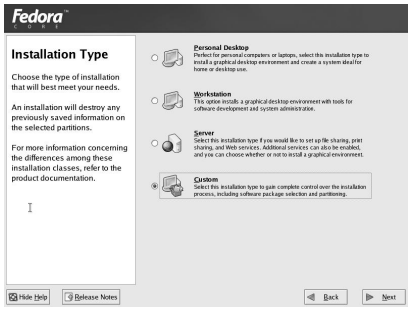


Figuur 3: Scherm om CD te testen

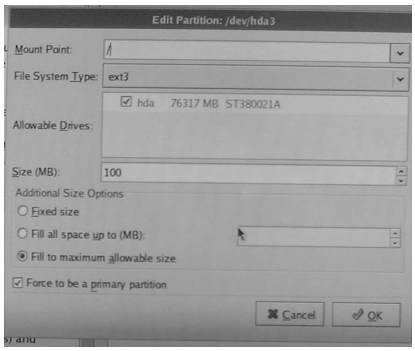
In Figuur 2 moeten gewoon op enter drukken om met de setup te starten. Dan krijgen we een tweede venster, Figuur 3, waar we de CD op fouten kunnen testen. Ook dit doen we. Daarna kunnen we verder gaan met het aanpassen van de instellingen.

Het volgende venster is het welkomstvenster en het venster erna is het venster om de taal te kiezen. Daarna krijgen we nog enkele vensters waarmee we het toetsenbord kunnen instellen. Dit moet natuurlijk ingesteld worden op “belgian be latin”.

Bij het venster van Figuur 4 kunnen we kiezen welk type van installatie we gaan installeren. De verschillende mogelijkheden die we hebben zijn: desktop, server, workstation, custom. Wij moeten met ons eindwerk een server opzetten dus we kiezen natuurlijk voor server.



Figuur 4: Het venster om het type van installatie te kiezen

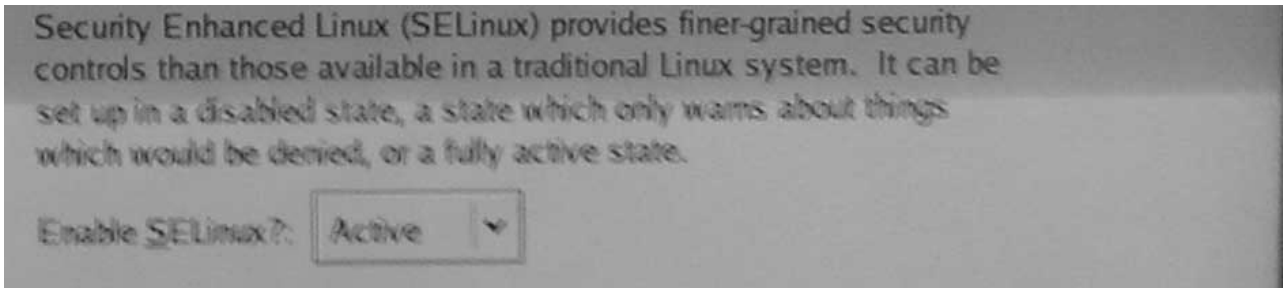


Figuur 5: Het venster om een partitie aan te passen

Het volgende wat we moeten doen is kiezen op welke partitie we Linux gaan installeren. Met `fdisk` hebben we partities aangemaakt. We installeren Linux op `/dev/hda3`. We moeten wel nog zeggen dat we het root filesysteem op deze partitie zetten. Dit kan je zien in Figuur 5.

Bij de volgende vensters kunnen we dual boot instellen. Dit hebben we nodig omdat we ook nog Windows op onze pc hebben staan. We veranderen de label van other naar Windows en zetten Windows als default zodat Windows standaard opstart en niet Linux.

De volgende vensters zorgen voor de netwerkinstellingen. Het venster dat daarna komt, in Figuur 6, is vrij belangrijk. Hier zien we SELinux optie staan. We moeten deze optie natuurlijk op active zetten, anders hebben we geen SELinux support.



Figuur 6: Het venster waar men SELinux kan activeren

Dan volgen er nog een paar instellingen die we moeten aanpassen: zoals de timezone instellen en het root paswoord instellen. Als dit alles gedaan is, kan de installatie beginnen. We klikken nu op next en de installatie begint. Tijdens de installatie moeten we wel enkele keren de cd vervangen. Na de installatie moeten we nog opnieuw opstarten en in het grub menu Fedora kiezen. Er volgen dan nog enkele instellingen die we moeten doen. Als we de eerste keer inloggen, komen we niet in kde terecht maar in fluxbox. Dit kunnen we veranderen door het commando

```
Switchdesktop kde
```

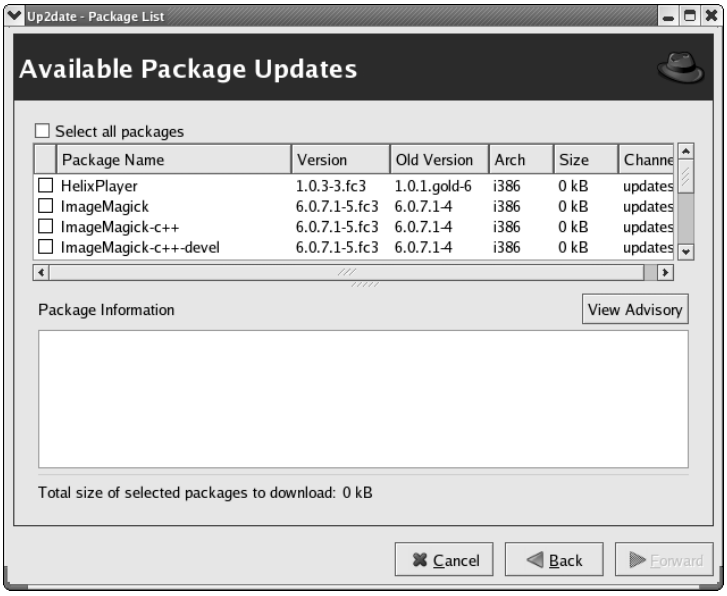
We gaan Fedora updaten met het volgende commando (zie Figuur 7):

```
Up2date
```

Met het volgende commando installeren we de grafische programma's voor SELinux (zie Hoofdstuk 6):

```
Up2date –install setools-gui
```

Nu is onze installatie compleet en kunnen we beginnen met het testen van deze distributie.



Figuur 7: Het venster van automatische updates

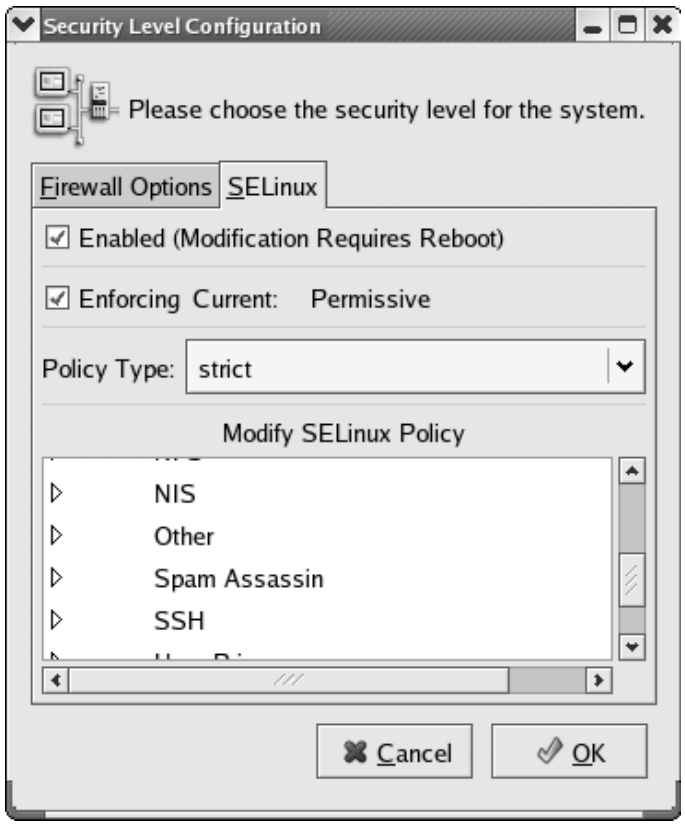
5.1.3 Testen van SELinux

In Figuur 8 en Figuur 9 kan men zien dat we bij Fedora twee modes hebben: we hebben de targeted mode en de strict mode.

De strict mode werkt gelijk elke ander distributie.

De targeted mode werkt iets anders dan de gewone distributies. Hierbij zij er daemons die draaien en die een beperkte beveiliging hebben. Deze daemons zijn: dhcpd, httpd (apache.te), named, nscd, ntpd, portmap, snmpd, squid, en syslogd. We kunnen dit ook allemaal via een menu aanpassen.

Figuur 8: Het targeted venster



Figuur 9: Het strict venster

In Figuur 7 kon men zien dat Fedora automatische updates heeft, zoals we die bij Windows kennen. Hierbij worden ook de policys up to date gehouden. Maar aan de strict mode wordt weinig aan gewerkt. Dat is ook begrijpelijk: men updatet meestal alleen de targeted mode.

De targeted mode is niet volledig veilig. We zien dat alleen de daemons die door de targeted mode ondersteund worden in hun eigen domain draaien als we ps -axZ (Z voor de security context) uitvoeren. Voor de rest draaien alle andere daemons gelijk ze dit zouden doen op een standaard Linux systeem.

De strict mode is hetzelfde als bij alle andere distributies. Het enige verschil is dat deze niet uptodate wordt gehouden. Ze wijzigen voorlopig alleen de targeted mode.

Waarom nu de targeted mode en niet de strict mode?
In het begin toen SELinux bij Fedora kwam was er alleen de strict mode. Hierin zaten nog veel fouten. Omdat de policy zo complex is, konden ze deze fouten er niet allemaal in een keer uit halen. Er waren ook nog veel programma's waar nog geen policy voor geschreven was. Daarom zijn ze van strategie veranderd en zijn ze stap voor stap gaan beveiligen: de targeted mode. De eerste programma's die ze hebben beveiligd zijn deze die het meest gevoelig zijn voor inbraken en misbruiken. Ze hebben ook een grafisch programma gemaakt, waarin je de beveiligingsinstellingen met een muisklik kan veranderen. Hiervoor hoef je de complexe regels van de policy van SELinux niet meer kennen. De programma's die ze nu aan het beveiligen zijn: ftp en mail daemons. Alle programma's die niet beveiligd zijn door de targeted mode worden in het `unconfined_t` domain uitgevoerd. Dit is een domain zonder beperkingen van SELinux. De daemons die wel beveiligd zijn, draaien in hun eigen domain. Bij het opstarten word sendmail opgestart in het `unconfined_t` domain, maar wanneer named opgestart wordt draait het in het `named_t` domain en wordt de juiste policy voor named gebruikt.
We vinden het persoonlijk wel raar dat ze sshd nog niet in de targeted mode hebben opgenomen.

De strict mode werkt ook wel in Fedora maar dan hebben we veel problemen om alles juist te configureren. Er zijn al wel enkele tools voorhanden die dit iets gemakkelijker maken. Dit is dezelfde tool als we bij de targeted mode ook hadden. We hebben alleen veel meer opties die we kunnen aanzetten. Deze tools worden besproken in hoofdstuk 6.

Ook hebben we bij Fedora Core 3 het commando `sestatus`. Voor meer uitleg over dit commando zie 5.4.5 Testen van SELinux.

5.2 Installeren van SELinux op Slackware 9.1

Omdat onze promotor het op prijs zou stellen dat we SELinux aan de praat zouden krijgen in Slackware zijn we hieraan begonnen. Voor deze distributie gaan we dan ook de volledige installatie beschrijven.

5.2.1 Afhalen van de nodige bestanden

Om SELinux te installeren, hebben we natuurlijk wel een aantal bestanden nodig. Het eerste wat we nodig hebben is een kernel met SELinux. De eerste keer hebben we de kernel 2.6.6 gebruikt. In het volgende hoofdstuk gebruiken we alle laatste nieuwe versies. Deze kernel kan je vinden op www.kernel.org. Ook hebben we een aantal progamma's nodig voor SELinux. We hebben deze packages gevonden op de site van Timothy Wood, een persoon die ook met SELinux in Slackware bezig is. Je kunt deze packages vinden op: http://www.diyab.net/SELinux.

5.2.2 Compileren van de kernel

Eerst moeten we de kernel uitpakken.

```
Mv Linux-2.6.6 /usr/src/
Cd /usr/src/
Tar xvzf Linux-2.6.6.tgz
```

```
Ln –s Linux-2.6.6 Linux
Cd Linux
Make mrproper
Make menuconfig
```

Nu krijgen we een menu te zien waar we verschillende kernel opties aan en uit kunnen zetten. Dit zijn de opties die belangrijk zijn voor SELinux.

bij filesystems ext2,3

```
extended attributes
security labels options
```

bij pseudo filesystems

```
/dev/pts extended attributes
/dev/pts security labels options
```

bij security

```
enable different security models
socket and networking security hooks
capabilities support
nsa SELinux support
nsa development support
nsa Linux boot parameter
```

Vergeet ook zeker de standaard dingen niet juist te zetten zoals de juiste processoropties en je netwerkkaart.

Nu sluiten we af, slaan we onze instellingen op en gaan we over tot het compileren van de kernel. Het eerste commando wat we typen is:

```
Make bzImage
```

Hiermee wordt de image van de kernel gemaakt. Daarna typen we:

```
Make modules
```

Dit commando maakt alle modules. Dit zullen er niet zo veel zijn omdat we meestal alles in de kernel compileren. Daarna volgt het commando:

```
Make modules_install
```

Dit commando zal er voor zorgen dat alle modules worden geïnstalleerd.

Nu is de kernel volledig gecompileerd. We hoeven enkel nog alle files op de juiste te plaats zetten en ons MBR (master boot record) veranderen. Dit doen we met de volgende commando's

```
Mv arch/i368/boot/bzImage /boot/bzImage-SELinux
Mv System.map /boot/System.map-SELinux
vi /etc/lilo.conf
```

Hier voegen we deze regels aan toe.

```
default=SELinux
image=/boot/bzImage-SELinux
root=/dev/hda1
label=SELinux
read-only
```

esc :wq (stoppen en wegschrijven van bestand)

Daarna voeren we nog gewoon `lilo` uit.

De kernel is nu klaar voor gebruik. Vooraleer we opnieuw opstarten, moeten we eerst nog enkele programma's installeren.

5.2.3 Installeren van de nodige SELinux programma's

Deze simpele packages kunnen we gewoon installeren met `upgradepkg` als er al een package geïnstalleerd is of met `installpkg` als de package nog niet geïnstalleerd is.

Deze packages moeten wel in een speciale volgorde worden geïnstalleerd. We beginnen met de packages van `glibc`.

Daarna:
`libSELinux`: dit is een library nodig voor SELinux
`checkpolicy`: compileert policy source naar binairy
`policycoreutils`: dit zijn programma's om de policy te loaden en te maken

Dan alle aangepaste progamma's:
`ssh`
`cron`
`coreutils`
`findutils`
`libuser`
`openssh`
`procps`
`shadow`
`sysvinit`
`util Linux`
`Pam` (zeker installeren anders kan je niet opstarten)

Ten slotte de policy zelf. We moeten een directory maken waar SELinux zijn bestanden kan zetten.

```
mkdir /selinux
```

Nu kunnen we opnieuw opstarten en kiezen voor de SELinux kernel.

```
Shutdown –r now
```

Inloggen met gebruikersnaam en paswoord

`Cd /etc/security/SELinux/src/policy`

`Make relabel` om het filesysteem helemaal te relabelen met SELinux attributes (zie werking van SELinux)

Nog eens herstarten. (`shutdown –r now`)

Als we nu inloggen, kunnen we kiezen in welke role we willen werken. Root kan in de `user_r` role of `sysadm_r` role werken.

5.2.4 Testen van SELinux

SELinux op Slackware staat nog niet heel ver. Er zijn weinig mensen bezig met het schrijven van een policy voor Slackware. In tegenstelling tot Gentoo en Fedora. We krijgen SELinux wel werkende maar er zijn nog maar weinig daemons die in het juiste domain draaien, deze zijn: `sshd, crond, klogd, syslogd, getty, init, kernel`. En de daemons die in het juiste domain draaien werken ook nog niet naar behoren. Ssh werkt heel goed in permissive mode: Als men met root inlogt via ssh dan veranderd de context naar `user_r`. Dit is gelijk het beschreven staat in `/etc/security/default_context`. In enforcing kunnen we gewoon niet inloggen. Ook andere daemons die beveiligd zouden moeten zijn werken nog niet goed. `Httpd` en `dhcpd` draaien zelfs nog niet in het juiste domain en werken dus nog niet. De policy werkt dus nog niet goed. SELinux werkt wel goed. Maar de juiste regels zijn nog niet toegevoegd in de policy.
Ook zijn er in Slackware nog niet echt programma's voorzien die het gebruik van SELinux vergemakkelijken. Er is maar een klein scriptje dat van enforcing naar permissive mode kan gaan. Dit programmaatje heet: `setenforce (0,1)`. Het is niet zoals bij Fedora waar we volledig grafisch de policy kunnen aanpassen.

5.3 Installeren van SELinux op Slackware 10.1

5.3.1 Afhalen van de nodige bestanden

Dit zijn nog testbestanden: ze zijn immers nog niet publiek gemaakt. Je kan al deze bestanden afhalen op deze url:
http://66.93.212.109/selinux
Voor de kernel gebruiken we nu de laatste versie 2.6.10 en de patch van de NSA site.

5.3.2 Compileren van de kernel

Het enige verschil met het compileren van de kernel is dat we deze kernel nog moeten patchen. Dit is niet moeilijk, we doen dit als volgt:

```
cp patch.gz /usr/src
gunzip patch.gz
cd Linux-2.6.10
patch -p1  < ../patch
```

We moeten extra opties aanzetten, deze zijn:

```
pseudo filesystems
virtual memory file
system, tmpfs EAs en tmpfs labels
```

Voor de rest blijft dit juist hetzelfde als bij Slackware 9.1.

5.3.3 Installeren van de SELinux programma's

Ook dit is weer hetzelfde als bij een Slackware 9.1. Alleen moeten we een programma zelf nog compileren en installeren.

We kunnen `u-dev` (userspace implementation van devfs) op veel plaatsen afhalen. Wij hebben het afgehaald op http://www.kernel.org. Voor de installatie gebruiken we de volgende commando's:

```
Tar xvzf ……
Cd …..
./configure
make
make install (checkinstall)
```

De installatie is voltooid; we hoeven enkel nog een beetje aan te passen. Bij `/etc/rc.d/rc.udev` moeten we een regel aanpassen: we zoeken naar

```
mount -n -t ramfs none $udev_root
```

en veranderen dit in:

```
mount -n -t tmpfs none $udev_root
```

Ook moeten we voor pam nog iets aanpassen.

In `/etc/securetty` moeten we deze regels nog toevoegen:

```
Vc/1
Vc/2
Vc/3
Vc/4
Vc/5
Vc/6
```

Anders kan root niet inloggen. `Pam` gebruikt blijkbaar de virtuele consoles om in te loggen. Als we dit vergeten, kunnen we dit nog altijd oplossen door via ssh in te loggen.

5.3.4 Testen van SELinux

Zoals bij Slackware 9.1 is het ook ongeveer bij Slackware 10.1. We hebben hier de nieuwste versies geprobeerd. Maar hier zijn we op veel problemen gebotst. De enforcing mode werkt nog totaal niet. In permissive draait alles redelijk goed. Maar als we naar enforcing mode gaan dan crashet alles. Er zitten dus nog zeer grote fouten in de packages of in de policy. De testen de we gedaan hebben zijn dus niet op Slackware 10.1 gebeurd omdat deze nog niet werkt.

Het algemene besluit van SELinux op Slackware is dat er nog te weinig support is. Er zijn ook maar heel weinig mensen bezig met de ontwikkeling van een policy voor Slackware. Wij hebben slechts 1 persoon gevonden die er echt mee bezig is. Voor grotere distributies zoals Gentoo en Fedora lopen de ontwikkelingen veel vlotter. Daar is zeker wekelijks een policy update.

5.4 Gentoo

5.4.1 De 3 stages

Het installeren van Gentoo is niet zo simpel. Dit is de moeilijkste distributie om werkende te krijgen. Het installeren bestaat uit 9 grote stappen.

- Opstarten in een omgeving om Gentoo te installeren
- Configureren van internetconnectie
- Partitioneren van de harde schijven
- Chrooten naar de nieuwe installatieomgeving
- Installeren van de basispackages die dezelfde zijn voor iedere Gentoo installatie
- Compileren van de nieuwe kernel

- Schrijven van de configuratiebestanden
- Installeren van de system tools
- Installeren van de boot loader

Vooraleer we met de installatie beginnen, geven we eerst een korte uitleg over de stages die Gentoo bezit. Bij Gentoo hebben we 3 stages.

Een stage1 installatie kan alleen wanneer je een internetverbinding hebt.
De voordelen:

- Je hebt de totale controle over de optimalisatie en over de build time voor je systeem. Bij deze stap wordt dus het meeste rekening gehouden met je systeem.
- Je kan veel aanpassen zodat alles optimaal werkt.
- Je leert meer bij over de innerlijke werking van Gentoo.

De nadelen:

- De installatie duurt lang.
- Als je niet van plan bent de instellingen te optimaliseren is het waarschijnlijk een verlies van tijd.
- Internetconnectie is vereist.

Stage2 laat je toe om het bootstrapproces over te slaan. Dit is goed als je tevreden bent met de standaardinstellingen die in de stage2 zitten. Ook deze stage kan je alleen maar gebruiken als je een werkende internetconnectie hebt.
De voordelen:

- Je kan het bootstrap proces overslaan.
- Stage2 is sneller dan stage1.
- Je kan je instellingen nog steeds aanpassen.

De nadelen:

- Je kan niet zoveel aanpassen als in stage1.
- Het is niet de snelste methode om Gentoo te installeren.
- Je moet de instellingen nemen van voor het bootstrappen.
- Internetconnectie is vereist.

Stage3 is de snelste manier om Gentoo te installeren. Het houdt wel in dat je tevreden moet zijn met de optimalisatie die gekozen is. We kiezen ook voor stage3 als we geen internetconnectie hebben.

De voordelen:

- Het is de snelste manier om Gentoo te installeren.
- Kan ook zonder een internetconnectie.

De nadelen:

- Je kan je basissysteem niet optimaliseren.
- Je leert niets bij.

Tot zover de verschillende stages. Wij zullen later kiezen voor stage2. Dit is sneller en toch kunnen we nog optimaliseren.
De volgende keuze die we moeten maken is welke cd we gaan downloaden om te beginnen met onze installatie. Hier hebben we de keuze tussen een gewone livecd en een minimal livecd.
Een gewone livecd is een cd die kan booten en die onze hardware grotendeels meteen herkent. Als deze cd opgestart is, zit je in een Gentoo omgeving, die klaar is om Gentoo te installeren. Op deze cd staan ook de verschillende stages. Het kan ook zijn dat de volledige software al op deze cd's staat zonder dat je nog iets van het internet moet afhalen.
Het verschil met een minimal livecd is dat op deze geen stages en geen software staan.
Wij kiezen voor een minimal livecd omdat onze software (voor SELinux) nog niet standaard op een cd staat. Deze is wel al te verkrijgen over de verschillende mirrors.

5.4.2 Starten van de installatie

We moeten natuurlijk werken met een livecd die SELinux ondersteunt. Als we deze cd opstarten zitten we rechtstreeks in SELinux. We gebruiken een aantal opstartopties.

`Boot: permissive dokeymap`

`Permissive`: om SELinux in permissive mode op te starten
`Dokeymap`: om onze toetsenbordinstellingen te kunnen aanpassen

Tijdens het opstarten zien we dat Gentoo alle hardware herkent. Dan krijgen we de root prompt.

Er zijn nu nog een paar dingen die we kunnen doen. Het eerste is natuurlijk ons toetsenbord juist instellen. Dit doen we met het commando:

```
Loadkeys be-latin1
```

Zo, nu hebben we een juiste toetsenbordindeling. Als Gentoo nog niet alles herkent dan kunnen we deze modules nog laden.

`Modprobe 8139too` : voor een Realtek netwerkkaart

Ook kunnen we onze harde schijven optimaliseren. Dit doen we met

```
Hdparm –tT /dev/hda
```

De parameters die we meegeven zijn om de harde schijf te testen. Om bijvoorbeeld dma aan te zetten gebruiken we deze regel:

```
Hdparm -d 1 /dev/hda
```

We kunnen ook nog users aanmaken voor tijdens de installatie. En de sshd daemon

```
/etc/init.d/sshd start
```

starten om andere mensen ons te laten helpen tijdens de installatie. Wij doen deze beide niet.

Ook kunnen we met links2 de handleiding op het internet volgen. Wij hebben dit gedaan op een aparte computer.

Bij het booten herkent Gentoo onze netwerkkaart. Onze pc krijgt een IP van een dhcp server op het netwerk. Mocht dit niet het geval zijn dan moeten we de

netwerkkaart nog configureren. Dit zullen we niet beschrijven: we verwijzen naar de informatie op het internet.

Nu kunnen we beginnen met het partitioneren van onze harde schijf. We beschrijven dit hier opnieuw omdat we voor Gentoo een aparte partitietabel nodig hebben. Dit doen we zoals eerder beschreven met fdisk.

We moeten uiteindelijk een partitietabel bekomen zoals in Tabel 1 is afgebeeld.

Tabel 1: De partitietabel

/dev/hda5	Ext2	32M	Boot partitie
/dev/hda6	Swap	512M	Swap partitie
/dev/hda7	Ext3	De rest van de harde schijf	Root partitie

De juiste partities zijn al gemaakt. We moeten alleen nog de juiste filesystems op deze partities installeren, zoals we in de bovenstaande tabel zien. Dit doen we met de commando's:

`Mke2fs /dev/hda5` : om ext2 te maken
`Mke2fs –j /dev/hda7` : om ext3 te maken, de j staat voor journal
`Mkswap /dev/hda6` : om swap te maken
`Swapon /dev/hda6` : om de swappartitie te activeren

Nu moeten we deze partities op de juiste plaats mounten. Dit doen we met de volgende commando's:

`Mount /dev/hda7 /mnt/Gentoo` : om de root directory te mounten
`Mkdir /mnt/Gentoo/boot` : om de boot directory te maken
`Mount /dev/hda5 /mnt/Gentoo/boot` : om de boot directory te mounten.

Nu moeten we nog de datum controleren. Dit doen we met het commando

```
Date
```

Als de datum niet klopt kunnen we deze aanpassen met:

```
Date 042516212004 25 april 2004 16.21u
```

We beginnen met het downloaden van een stage. We hebben eerder besloten om stage2 te gebruiken. Deze moeten we nu nog afhalen. Eerst gaan we naar de directory waar deze file moet komen te staan.

```
Cd /mnt/Gentoo
```

Nu gaan we surfen op het internet om onze stage2 af te halen.

```
Links2 http://www.Gentoo.org/main/en/mirrors.xml
```

Daarna kiezen we ergens een mirror in België (omdat dit sneller gaat). De stages staan bij `/experimental/x86/hardened/stages`. We hebben deze van SELinux nodig.

Voor we de stage uitpakken, gaan we deze eerst controleren.

```
md5sum -c stage1-x86-SELinux-2004.2.tar.bz2.md5
stage: ok
```

We weten dat de stage goed is en we pakken deze uit.

```
tar -xvjpf stage?-*.tar.bz2
```

We kunnen nog wat opties aanpassen voor een optimale compilatie.
Deze opties staan allemaal in

```
/mnt/Gentoo/etc/make.conf
```

Er staat een heel goed voorbeeld in `/mnt/Gentoo/etc/make.conf.example`

Onze `make.conf` zag er uiteindelijk zo uit:

```
FLAGS=”-O2 –mcpu=i686 –fomit-frame-pointer –pipe”
CHOST=”i686-pc-Linux-gnu”
CXXFLAGS=”${CFLAGS}”
MAKEOPTS="-j2"
USE=”pic hardened”
```

`-pipe`: zorgt ervoor dat pipes gebruikt worden in plaats van tijdelijke files voor communicatie tussen verschillende processen tijdens de compilatie.
`Fomit-frame-pointer`: houd frame pointers niet bij in het register voor functies die het niet nodig hebben.
`MAKEOPTS` is voor het aantal parallelle processen tijdens compilatie. In ons geval 2.

`Pic` en `hardened` hebben we nodig voor SELinux.

Eerst moeten we ook nog onze DNS-info kopiëren.

```
Cp –L /et c/resolv.conf  /mnt/Gentoo/etc/resolv.conf
```

`–L` zorgt ervoor dat we geen symbolische link mee kopiëren.

We moeten de directorys proc en selinux nog mounten:

```
mount -t proc none /mnt/gentoo/proc
mount -t selinuxfs none /mnt/gentoo/selinux
```

Nu gaan we omschakelen naar de nieuwe omgeving.

`Chroot /mnt/gentoo /bin/bash` :	/ komt nu overeen met `/mnt/gentoo`
`Env-update`:	maakt de omgevingsvariabelen aan
`Source /etc/profile` :	het laden van deze variabelen in het geheugen

De volgende stap is het downloaden van de portage tree van het internet.

`Emerge sync` (eventueel `emerge web-rsync` als de firewall het niet toelaat)

Als we een stage1 hadden moesten we het systeem eerst nog bootstrappen. Dit zouden we doen met:

`cd /usr/portage`
`scripts/bootstrap.sh –f:` om de broncode af te halen
`cd /usr/portage`
`scripts/bootstrap.sh:` om echt te bootstrappen.

Wij gebruiken stage2, dus we kunnen deze stap overslaan. We gaan meteen door met het maken van de systeem packages.

`emerge --pretend system | less:` om te kijken wat er wordt afgehaald
`emerge --fetchonly system:` om de bron code te downloaden
`emerge system:` om het systeem te compileren

Deze vorige stappen kunnen een tijdje duren. Na deze stappen gaan we verder met het configureren van ons systeem.

We gaan eerst ook nog onze tijdzone goed zetten. Deze zones vinden we bij `/usr/share/zoneinfo.`
We maken een symbolische links naar `/etc/localtime`

```
ln -sf /usr/share/zoneinfo/Europe/Brussels  /etc/localtime
```

5.4.3 Compileren van de kernel

Nu moeten we een kernel afhalen en de juiste links leggen.

```
Emerge hardened-sources
Rm /usr/src/Linux
```

```
cd /usr/src
```

`ln -s Linux-2.6.7-hardened-r6 Linux` : om Linux naar de huidige versie te laten wijzen.

Nu gaan we de kernel compileren

```
Cd /usr/src/Linux
Make menuconfig
```

We krijgen nu een menu waar we verschillende opties kunnen aanpassen. De opties die we zeker moeten activeren voor SELinux zijn:

Onder "Code maturity level options"

```
[*] Prompt for development and/or incomplete code/drivers
```

Onder "General setup"

```
[*] Auditing support
```

Onder "File systems"

```
<*> Second extended fs support (If using ext2)
[*]   Ext2 extended attributes
[ ]     Ext2 POSIX Access Control Lists
[*]     Ext2 Security Labels
<*> Ext3 journalling file system support (If using ext3)
[*]   Ext3 extended attributes
[ ]     Ext3 POSIX Access Control Lists
[*]     Ext3 security labels
<*> JFS filesystem support (If using JFS)
[ ]   JFS POSIX Access Control Lists
[*]   JFS Security Labels
[ ]   JFS debugging
[ ]   JFS statistics
<*> XFS filesystem support (If using XFS)
[ ]   Realtime support (EXPERIMENTAL)
[ ]   Quota support
```

```
[ ]   ACL support
[*]   Security Labels

[*] /proc file system support
[ ] /dev file system support (EXPERIMENTAL)
[*] /dev/pts file system for Unix98 PTYs (This option does not
appear in 2.6, it is always on)
[*]   /dev/pts Extended Attributes
[*]     /dev/pts Security Labels
[*] Virtual memory file system support (former shm fs)
[*]   tmpfs Extended Attributes
[*]     tmpfs Security Labels

Onder "Security options"
[*] Enable different security models
[*] Socket and Networking Security Hooks
<*> Capabilities Support
[*] NSA SELinux Support
[ ]   NSA SELinux boot parameter
[ ]   NSA SELinux runtime disable
[*]   NSA SELinux Development Support
[ ]   NSA SELinux AVC Statistics
[ ]   NSA SELinux MLS policy (EXPERIMENTAL)
```

Er zijn nog een aantal dingen die we zeker niet mogen vergeten te activeren in de kernel zoals processortype, netwerkkaart, … Dan slaan we onze wijzigingen op en gaan we onze kernel compileren.

```
make && make modules_install
```

Dit duurt een tijdje.

Vervolgens moeten we nog de juiste bestanden op de juiste plaats zetten.

```
cp arch/i386/boot/bzImage /boot/bzImage-2.6.11
```

```
cp System.map /boot/System.map-2.6.11
cp .config /boot/config-2.6.11
```

We kunnen nog andere extra modules installeren. Onze pc heeft een geforce grafische kaart. Deze kunnen we installeren met:

```
emerge nvidia-kernel
```

We kunnen onze modules automatisch laten laden door ze in `/etc/modules/autload.d/kernel-2.6` toe te voegen. Nu voeren we dit commando uit:

```
Modules-update
```

5.4.4 Voortzetten van de installatie

Om onze filesystems bij het opstarten gemount te krijgen, moeten we onze `/etc/fstab` file nog aanpassen. Onze `fstab` file ziet er uiteindelijk uit zoals hieronder.

```
/dev/hda5      /boot     ext2          noauto,noatime      1 2
/dev/hda6      none      swap          sw                  0 0
/dev/hda7      /         ext3          noatime             0 1

none        /proc    proc             defaults            0 0
none       /dev/shm  tmpfs            defaults            0 0
none       /dev/pts  devpts           gid=5,mode=620      0 0
none       /selinux  selinuxfs        defaults            0 0

/dev/hdc  /mnt/cdrom     auto      noauto,user    0 0
/dev/hdd  /mnt/cdrom2    auto      noauto,user    0 0
```

We gaan nu ons netwerk configureren.
Dit commando dient om onze netwerknamen in te stellen.

```
echo tux > /etc/hostname
echo homenetwork > /etc/dnsdomainname
echo nis.homenetwork > /etc/nisdomainname
```

Nu moeten we de connectie nog verder configureren. We moeten gewoon het bestand `/etc/conf.d/net` bewerken.
Dit zijn de regels die we nodig hadden om onze connectie werkende te krijgen.

```
Iface_eth0=”10.0.0.5 broadcast 10.255.255.255 netmask
255.255.255.0”
Netmask_eth0=”255.255.255.0 255.255.255.0”
Gateway=”eth0/10.0.0.1”
```

`rc-update add net.eth0 default` : deze regel dient om het netwerk te starten bij het booten.

We kunnen onze netwerkinformatie ook in het bestand `/etc/hosts` zetten. Zo moeten we geen ip-adressen meer gebruiken, maar kunnen we gewoon namen gebruiken.

Ook moeten we niet vergeten om het root paswoord goed te zetten. Dit is niet moeilijk. Als root moeten we gewoon het commando `passwd` typen. Dan kunnen we tweemaal het nieuwe paswoord ingeven.

Ook is er nog een heel belangrijke file die we moeten aanpassen. `/etc/rc.conf`
Het belangrijkste is onze toetsenbordinstelling. Deze moeten we weer zetten op

```
be-latin1.
```

Nu moeten we nog een system logger installeren. Dit kunnen we met twee simpele commando’s.

```
emerge metalog
rc-update add metalog default.
```

Ook vixie-cron gaan we installeren. Dit kunnen we op dezelfde manier doen.

```
emerge vixie-cron
rc-update add vixie-cron default
```

We hebben nog wat filesystem tools nodig.

```
emerge xfsprogs      voor xfs filesystem
emerge jfsutils      voor jfs filesystem
```

De volgende stap is en bootloader installeren zodat we bij het opstarten kunnen kiezen tussen Linux of Windows. We hebben keuze uit verschillende bootloaders. Wij hebben ervaring met lilo dus hebben we voor deze gekozen. Het afhalen van lilo gaat bij Gentoo vrij simpel. We moeten deze package weer gewoon installeren met het commando:

```
emerge --usepkg lilo
```

Dan moeten we de config file van lilo natuurlijk nog aanpassen.

```
Nano –w /etc/lilo.conf

boot=/dev/hda              # installeer lilo in de mbr
prompt                     # geef de gebruiker de kans een
                             keuze te maken
timeout=50                 # wacht 5 seconden vooralleer het
                           default besturingssysteem te booten
default=windows             # als de tijd is verstreken boot
hij deze
vga=792                    # om het scherm goed te zetten zie
                                Tabel 2
image=/boot/bzImage-2.6.11 # plaats waar onze image staat
label=Gentoo              # naam van deze sectie
 read-only                 # start met read-only voor root
```

```
 root=/dev/hda7          # plaats van het root filesystem
 append="Gentoo=nodevfs" # deze optie is nodig voor SELinux
# het volgende stuk is voor als je dual boot wenst.
other=/dev/hda1
  label=windows
```

Tabel 2: Verschillende resoluties en kleurdieptes

	640x480	800x600	1024x768	1280x1024
8bpp	769	771	773	775
16bpp	785	788	791	794
32bpp	786	789	792	795

Nu is onze config file geschreven. Nu moeten we dit nog in het master boot record (mbr) laden. Dit doen we met:

```
/sbin/lilo
```

Nu moeten we de policy voor SELinux nog installeren en relabelen.

```
cd /etc/security/SELinux/src/policy/
make load
make chroot_relabel
```

Nu gaan we uit onze chroot omgeving en gaan we alles unmounten en rebooten.

```
exit
umount /mnt/Gentoo/boot /mnt/Gentoo/proc /mnt/Gentoo/SELinux
/mnt/Gentoo
reboot
```

Het enige wat we nu nog moeten doen is het echte filesystem relabelen.

```
cd /etc/security/SELinux/src/policy
make relabel
```

De installatie is voltooid. We hebben wel nog geen grafische omgeving zoals kde of x geïnstalleerd maar dit is niet zo moeilijk:

```
Emerge xorg-x11
```

5.4.5 Testen van SELinux

Nu volgt een korte beschrijving over SELinux op Gentoo.
Al de files zijn dezelfde als bij andere distributies. Er zijn alleen een paar extra dingen bij Gentoo. Zoals het commando:

```
Sestatus –v
```

Dat geeft deze output:

```
SELinux status:         enabled
SELinuxfs mount:        /SELinux
Current mode:           permissive
Policy version:         18

Policy booleans:
secure_mode             inactive
ssh_sysadm_login        inactive
user_ping               inactive

Process contexts:
Current context:        root:sysadm_r:sysadm_t
Init context:           system_u:system_r:init_t
/sbin/agetty            system_u:system_r:getty_t
/usr/sbin/sshd          system_u:system_r:sshd_t

File contexts:
Controlling term:       root:object_r:sysadm_devpts_t
/sbin/init              system_u:object_r:init_exec_t
/sbin/agetty            system_u:object_r:getty_exec_t
```

```
/bin/login              system_u:object_r:login_exec_t
/sbin/rc                system_u:object_r:initrc_exec_t
/sbin/runscript.sh      system_u:object_r:initrc_exec_t
/usr/sbin/sshd          system_u:object_r:sshd_exec_t
/sbin/unix_chkpwd       system_u:object_r:chkpwd_exec_t
/etc/passwd             system_u:object_r:etc_t
/etc/shadow             system_u:object_r:shadow_t
/bin/sh                 system_u:object_r:bin_t ->
system_u:object_r:shell_exec_t
/bin/bash               system_u:object_r:shell_exec_t
/bin/sash               system_u:object_r:shell_exec_t
/usr/bin/newrole        system_u:object_r:newrole_exec_t
/lib/libc.so.6          system_u:object_r:lib_t ->
system_u:object_r:shlib_t
/lib/ld-Linux.so.2      system_u:object_r:lib_t ->
system_u:object_r:shlib_t
```

We kunnen deze waardes ook op een simpele manier aanpassen. Met het commando

```
togglesebool user_ping
```

of

```
setsebool user_ping 0
```

Het eerste stuk is makkelijk te begrijpen. Dit laat gewoon de status van SELinux zien. Het tweede stuk geeft de status weer van de policy booleans. Het derde stuk gaat over de context van de processen die draaien. Het laatste stuk gaat over de file context van enkele belangrijke files.

We kunnen ook makkelijker een stuk van het filesystem relabellen. Dit is een hele taak bij andere distributies. We kunnen dit met het commando:

```
rlpkg pam-login sash
```

Dit zijn zowat de belangrijkste verschillen met andere distributies waarop we SELinux hebben getest. De policy verschilt natuurlijk. De policy van Gentoo is al goed

ontwikkeld. Er zijn al heel wat daemons die in het juiste domain draaien. Ssh, httpd, dhcpd, crond, … . Deze die in het juiste domain draaien die werken ook. Niet zoals bij Slackware waar sommige daemons nog niet goed functioneren terwijl ze wel in het juiste domain draaien. We kunnen bij Gentoo naar enforcing overschakelen en de daemons die in het juiste domain draaien zullen dan ook nog juist functioneren. We kunnen SELinux in Gentoo ook makkelijk up to date houden. Met de use flags geven we te kennen dat we SELinux gebruiken.

```
USE=”pic hardened”
```

Als we ons systeem willen updaten dan kunnen we dat met dit commando

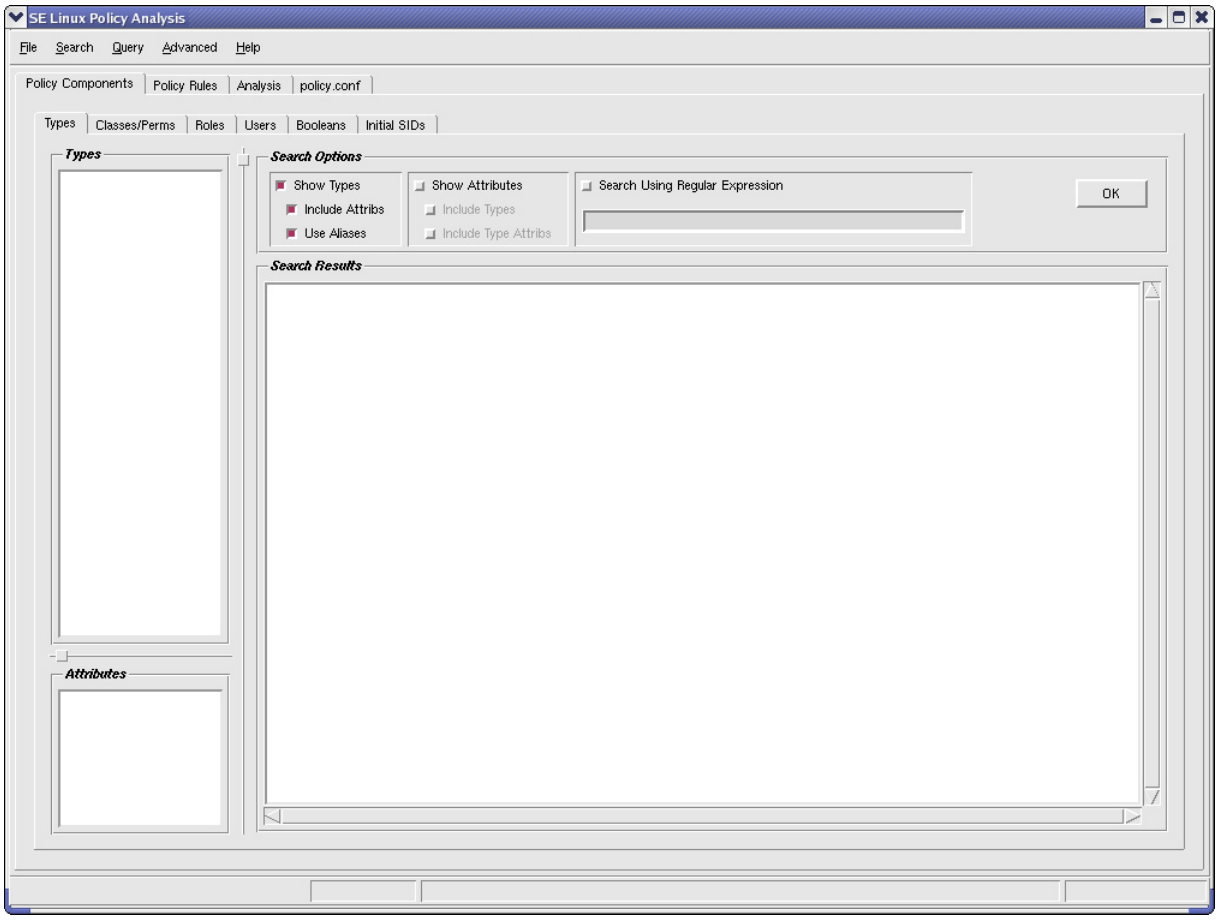
```
emerge --update --deep --newuse world
```

6 Grafische tools voor SELinux

De ontwikkeling van SELinux is nu vlot aan het lopen. Er zijn al verschillende grafische tools voorhanden die helpen bij het schrijven en begrijpen van de policy. In dit hoofdstuk gaan we deze tools beschrijven.

6.1 Apol

De meest uitgebreide tool van deze tools is apol (SELinux policy analysis). Hieronder vind je een screenshot van apol.



Figuur 10: Het startvenster van apol

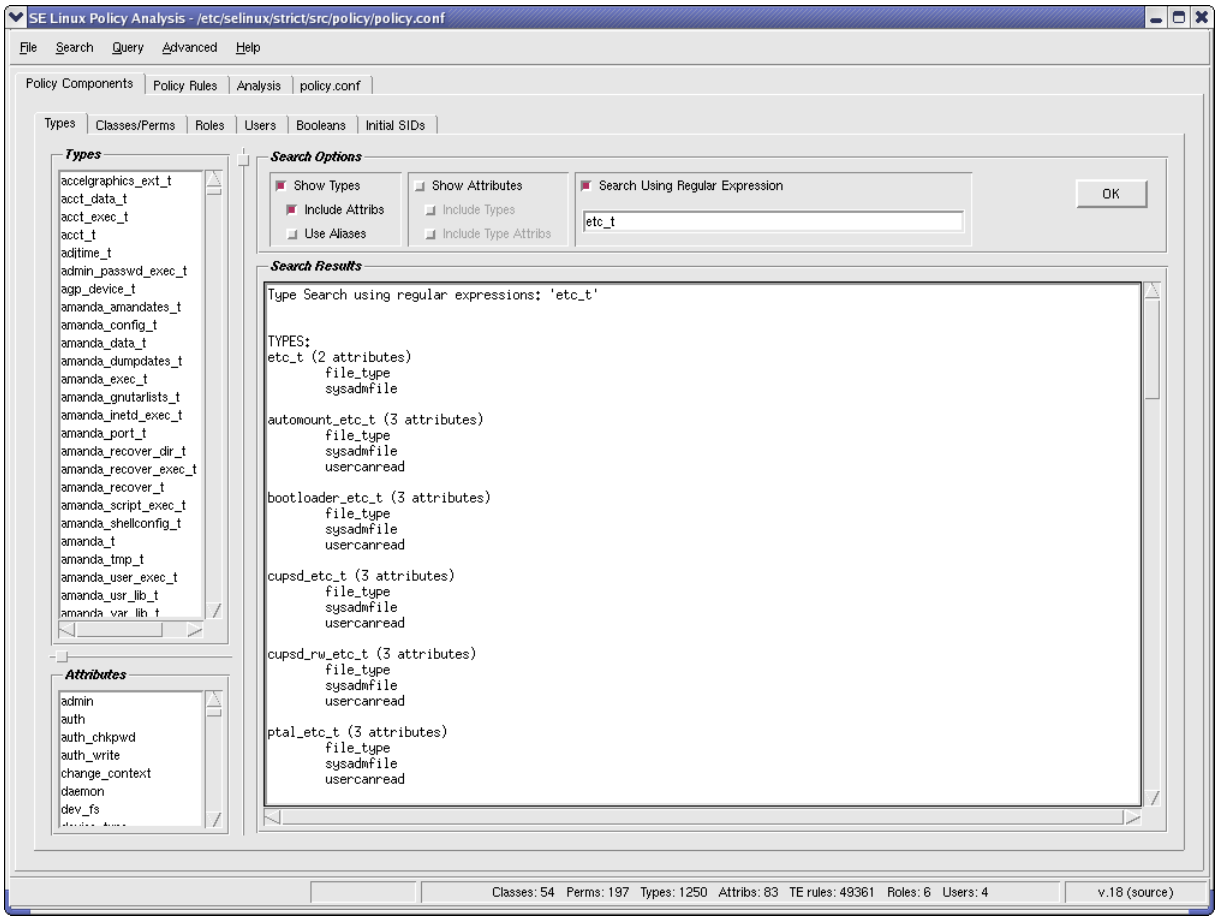
De eerste stap is apol openen. Daarna moeten we een policy openen. Omdat we deze tool in Fedora Core 3 getest hebben, kiezen we voor de stricte policy. Deze gecompileerde vorm bevindt zich in `/etc/selinux/strict/policy/policy.18`. We kunnen ook de `policy.conf`

file openen, deze bevindt zich in `/etc/selinux/strict/src/policy/policy.conf`. Als we deze file openen worden alle vensters in apol gevuld met informatie. We gaan hieronder elk tabblad beschrijven.

We hebben boven drie tabbladen met verschillende tabbladen eronder. De bovenste drie tabbladen zijn: policy components, policy rules en analysis. Als we de policy.conf file openen hebben we nog een extra tabblad policy.conf

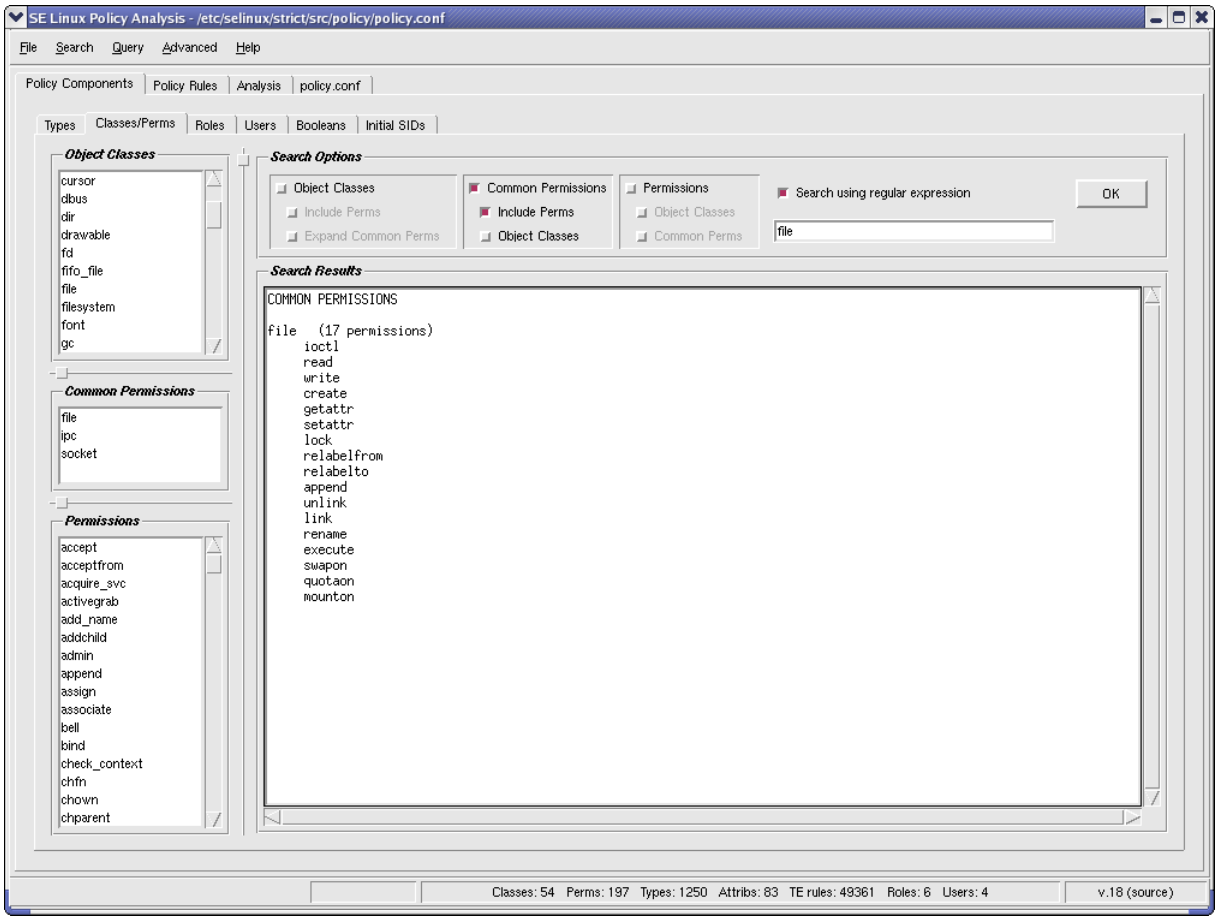
Het eerste tabblad policy components is onderverdeeld in 6 tabbladen. Deze bevatten alle policy componenten.

Het eerste tabblad beschrijft alle mogelijke types die we hebben in onze policy. Hier hebben we verschillende mogelijkheden om de types met hun attributen weer te geven. Dit is makkelijk als we ergens in een policy file een type tegen komen en we vinden niet direct welke attributen dit type heeft. Neem nu bijvoorbeeld dat we willen weten welke attributen het type `etc_t` heeft. We weten natuurlijk dat dit `file_type` en `sysadm_file` zijn. Maar dit gaan we nu eens testen met apol. We vinken show types en include attributes aan, dan vinken we ook nog het zoekveld aan en typen `etc_t` en vervolgens klikken we op ok.
We krijgen nu alle files die eindigen op `etc_t` met vanboven het type `etc_t`. Zoals verwacht zien we dat `etc_t` twee attributen heeft. Dit kunnen we in Figuur 11 zien.



Figuur 11: De types en attributen met etc_t via apol

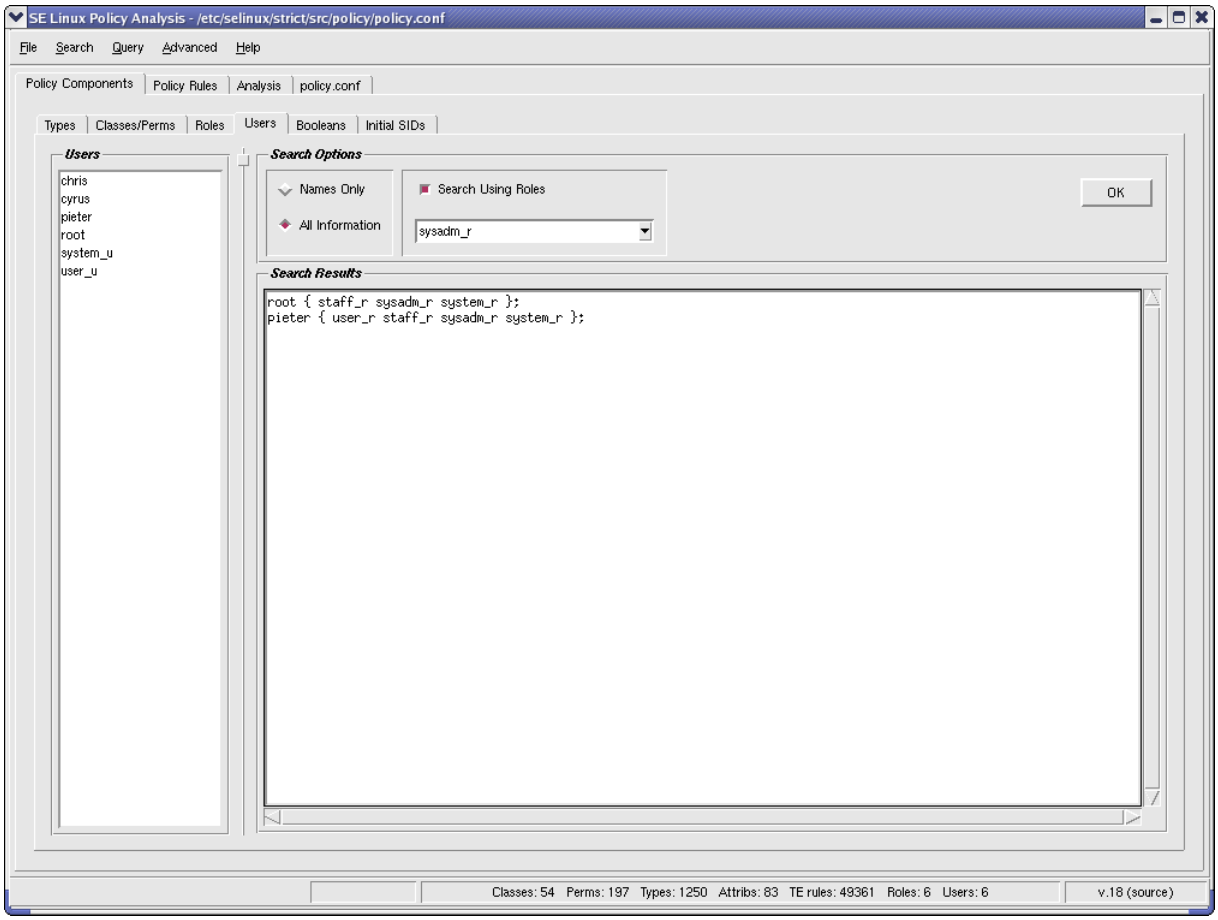
Het tweede tabblad slaat op de classes en permissies. Ook hier heeft men weer een uitgebreid zoeksysteem. Als we nu bijvoorbeeld willen weten welke permissies er allemaal mogelijk zijn op het object file dan kunnen we dit zoeken in dit tabblad. We vinken `common permissions` en `include perms` aan en vervolgens klikken we weer op het zoekveld en typen file in en klikken op ok. Vervolgens krijgen we zoals in Figuur 12 een lijst met de mogelijke permissies die van toepassing zijn op het object file.



Figuur 12: Toelatingen op het object file

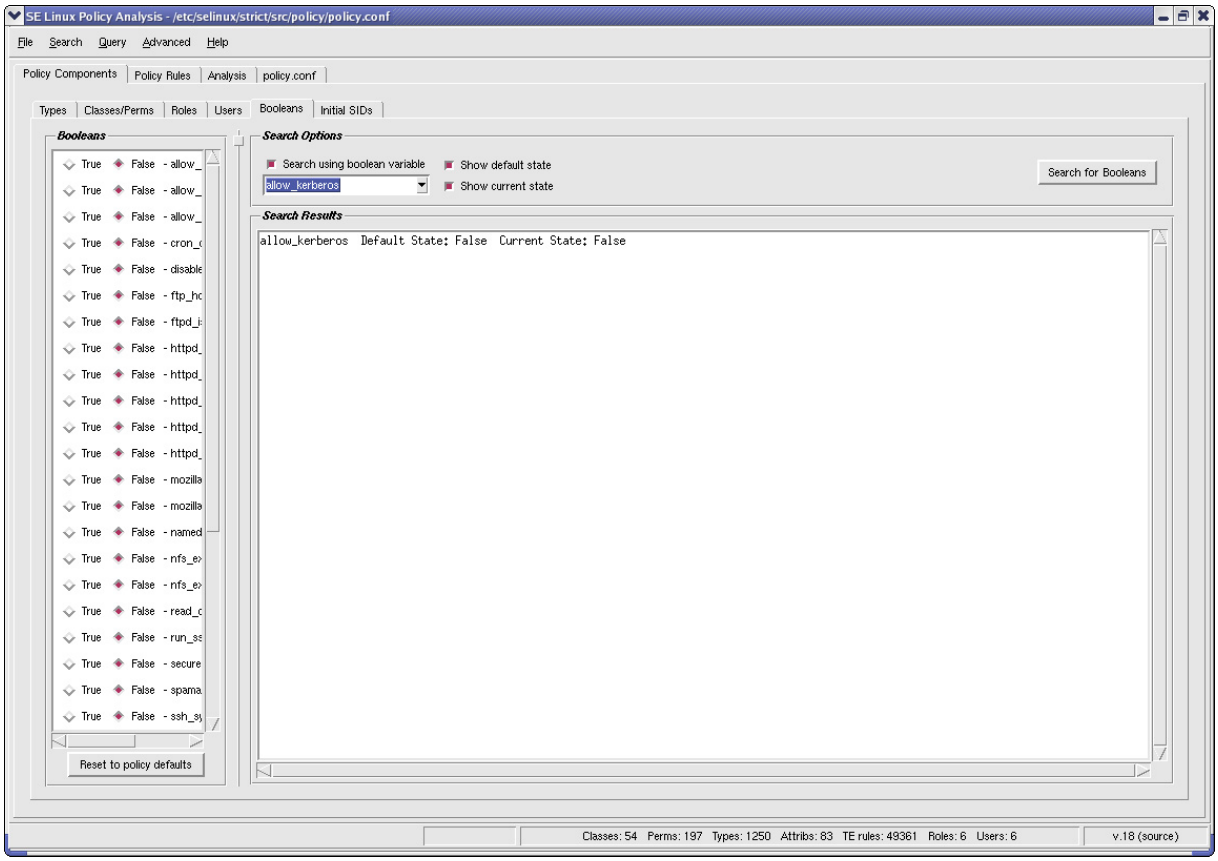
Het derde tabblad slaat op de verschillende rollen die er in onze policy zijn en welke rollen aan welke types kunnen. Dit tabblad werkt hetzelfde als de twee vorige tabbladen.

Het vierde tabblad geeft de verschillende users in de policy weer en welke rollen ze bezitten. Zoals men in Figuur 13 kan zien, zien we dat de root user en pieter in de `sysadm_r` rol kunnen komen. We zien ook dat Chris alleen maar de `user_r` rol bezit en dus geen speciale rechten heeft.



Figuur 13: Users met de sysadm_r rol

Het vijfde tabblad vindt men de booleans. Gentoo ondersteunt ook booleans. Alleen zijn er bij Fedora meer booleans. Zoals men in Figuur 14 kan zien is ook hier weer een zoeksysteem aanwezig. We kunnen de waardes wel niet wijzigen, we kunnen enkel kijken of ze geactiveerd zijn.

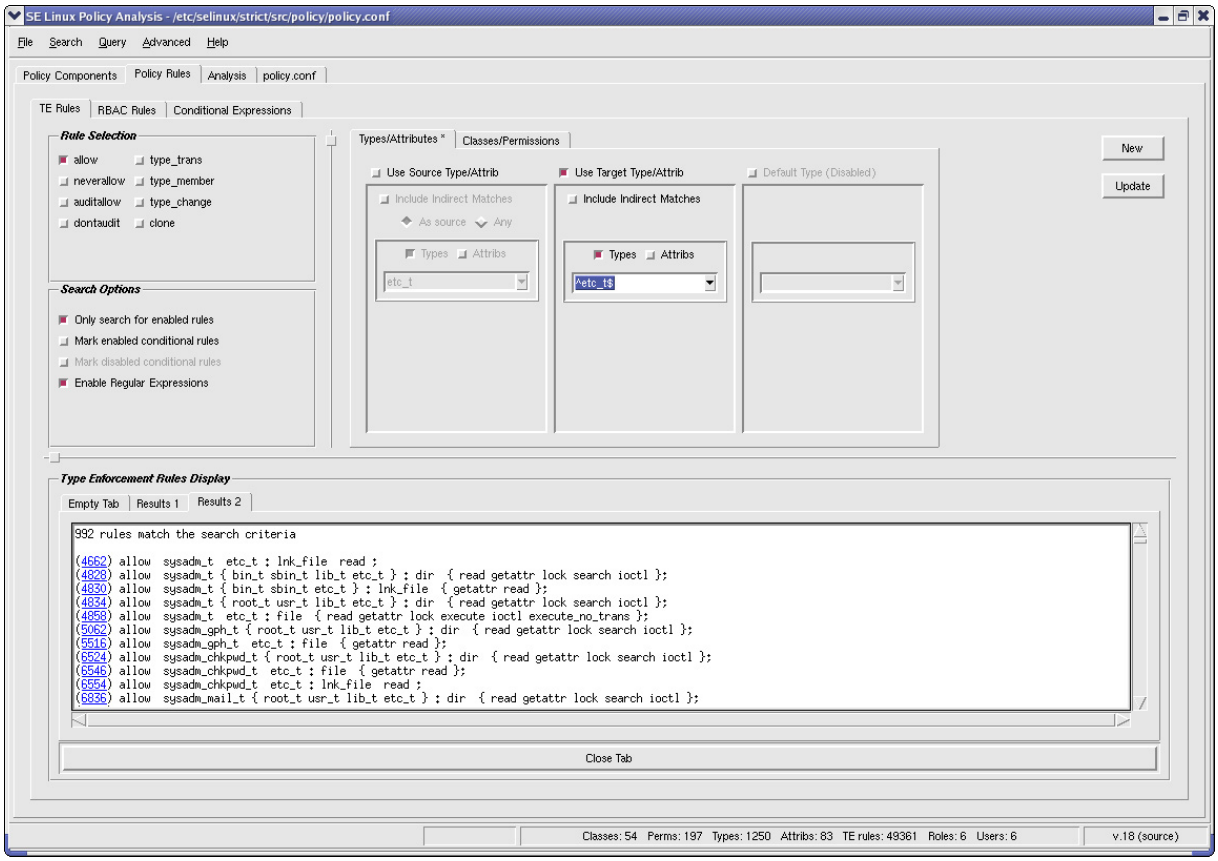


Figuur 14: De booleans

Het laatste tabblad geeft de initial sids weer. Deze gaan we normaal nooit wijzigen.

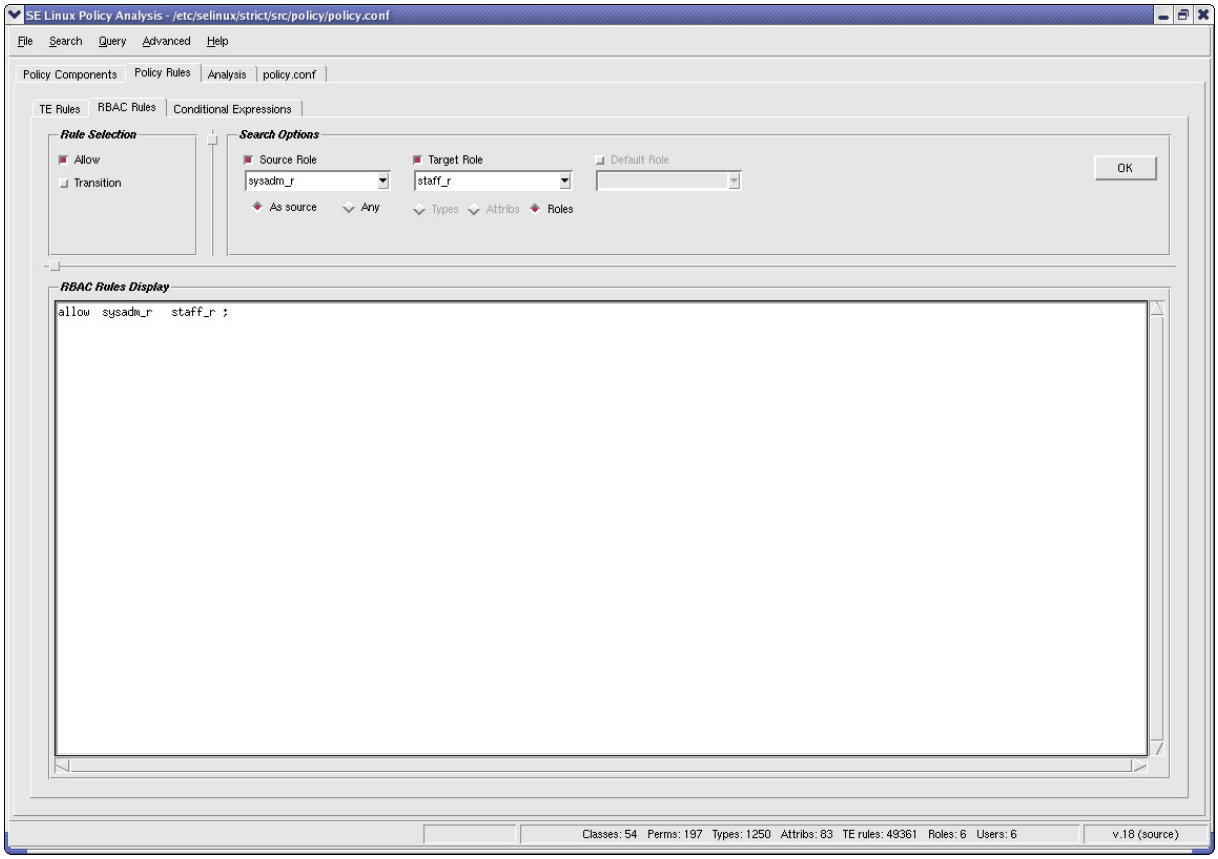
Dat waren de policy components. Nu gaan we door met **het tweede tabblad** namelijk de policy regels. Dit tabblad is eigenlijk het meest interessante. We hebben weer drie tabbladen in de policy regels. De te regels, de rbac regels en de conditional expressions.

Eerst gaan we het *tabblad van de te regels* kort beschrijven. In Figuur 15 hebben we gezocht op allow regels die betrekking hebben op `etc_t`.



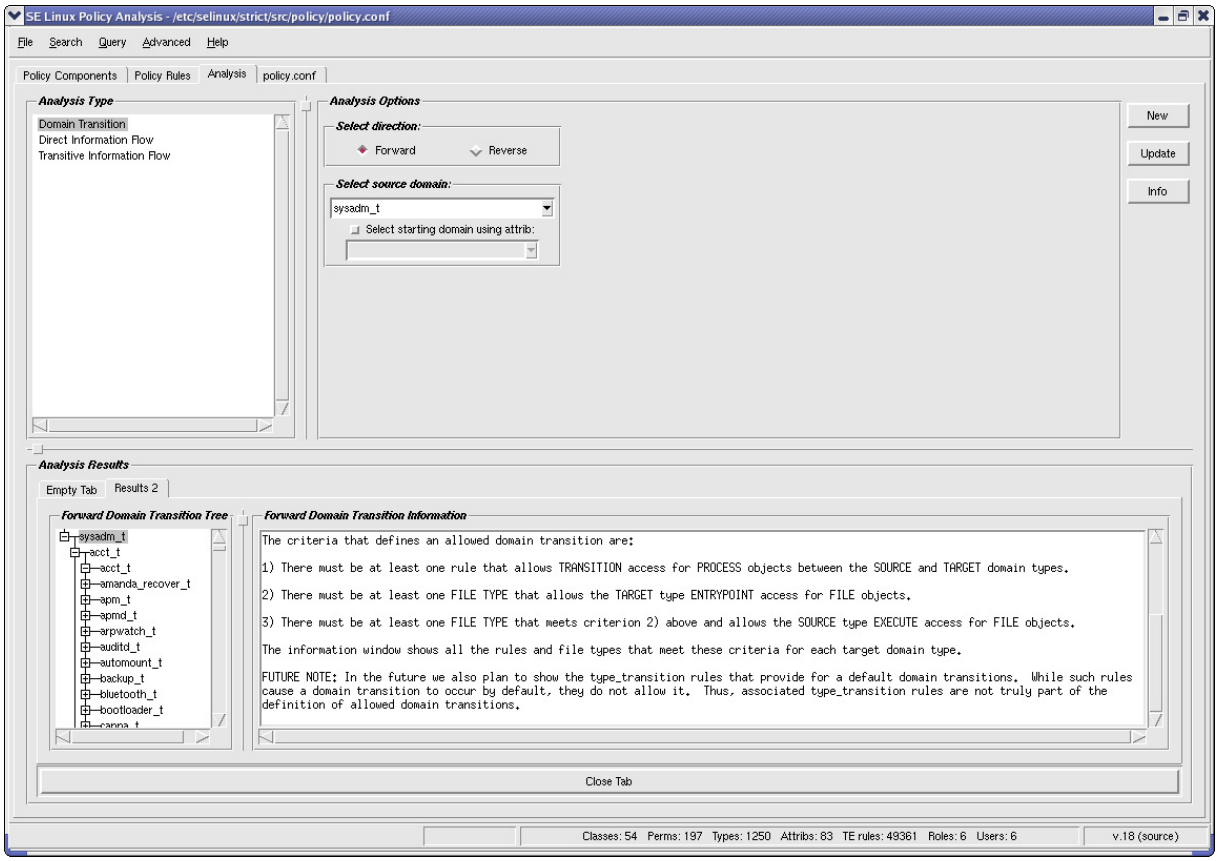
Figuur 15: Regels met betrekking op het type etc_t

In Figuur 16 zien we dat we op dezelfde manier kunnen zoeken op **rbac regels**.



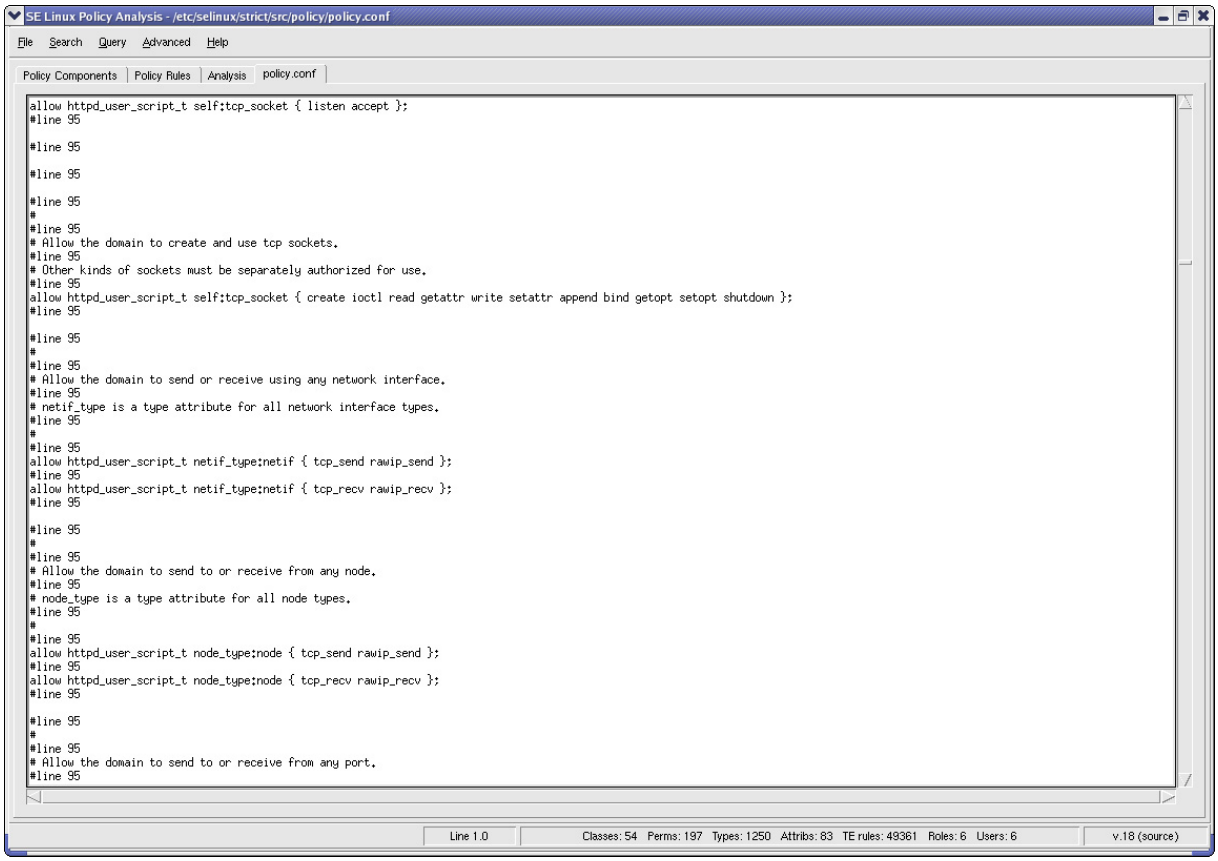
Figuur 16: RBAC regels van sysadm_r naar staff_f

Het derde hoofdtabblad noemt analysis. Dit is ook een nuttig tabblad. Dit geeft weer welke transities er allemaal mogelijk zijn. Als we willen weten in welke domains `sysadm_t` kan komen moeten we dit ingeven als starting domain. Dan krijgen we een lijst met alle domains waar `sysadm_t` in kan komen. Maar dat is nog niet alles. De domainen waar `sysadm_t` in kan komen kunnen op hun beurt weer naar een ander domain overgaan. Ook dit is bij apol duidelijk zichtbaar.
We kunnen ook in de omgekeerde richting zoeken.



Figuur 17: Domeinen waar sysadm_t naar kan overgaan

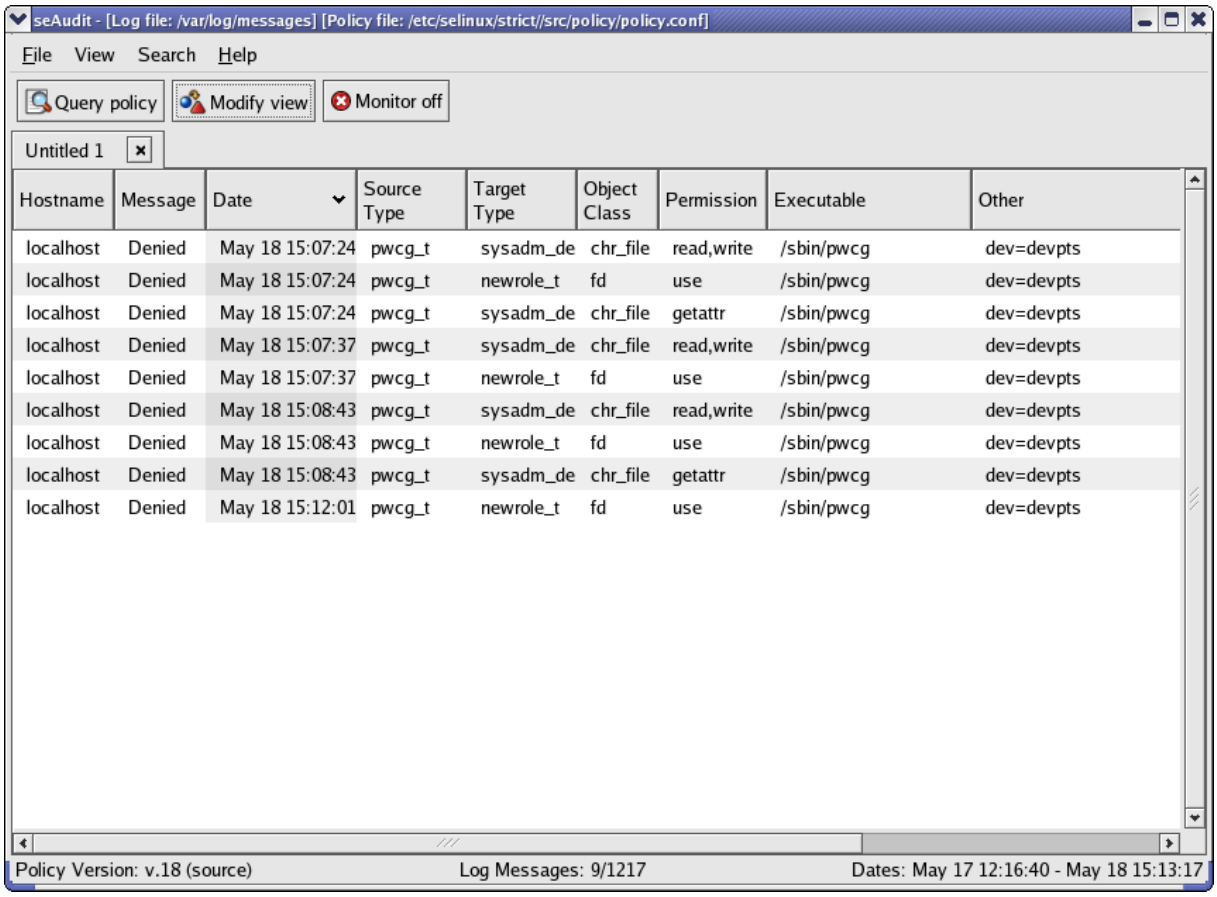
Het laatste tabblad geeft gewoon de policy.conf weer. Dit ziet men in Figuur 18.



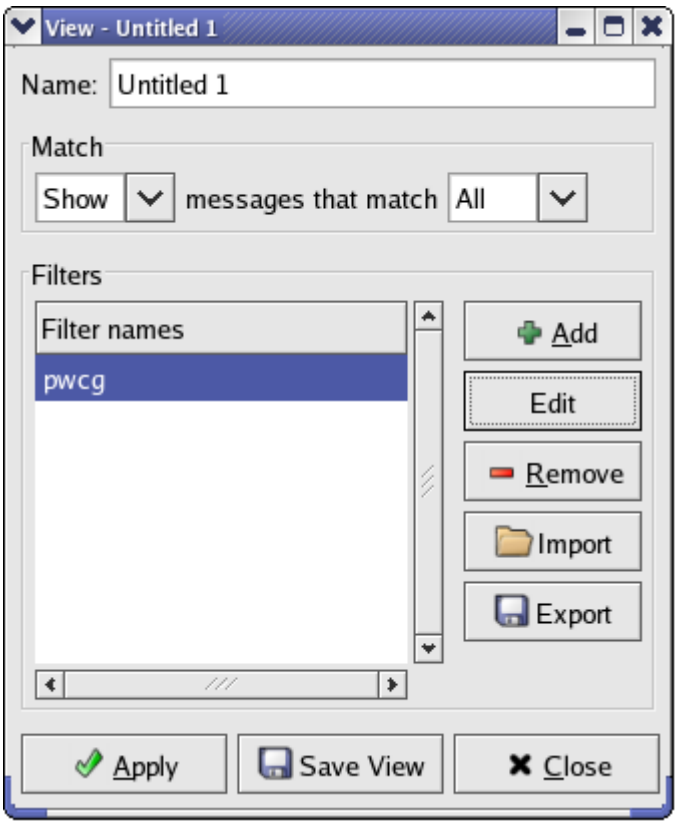
Figuur 18: De policy.conf file

6.2 Seaudit

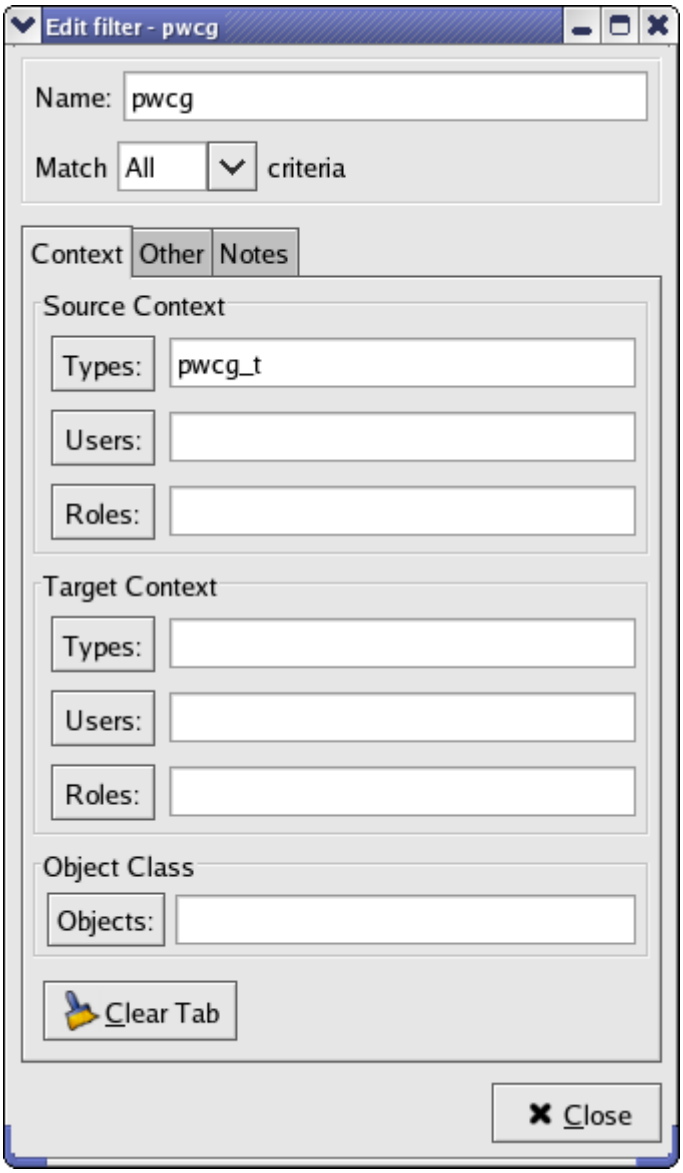
De volgende tool die we gaan beschrijven is seaudit. Deze tool maakt het makkelijker om de errors uit de policy te filteren. We kunnen zien hoe dit gebeurd in de onderstaande figuren. We moeten eerst een filter toevoegen (Figuur 20), deze instellen (Figuur 21) en deze vervolgens toepassen op de errors. In Figuur 19 ziet men alle errors met betrekking tot `pwcg_t`.



Figuur 19: Errors met betrekking tot pwcg_t

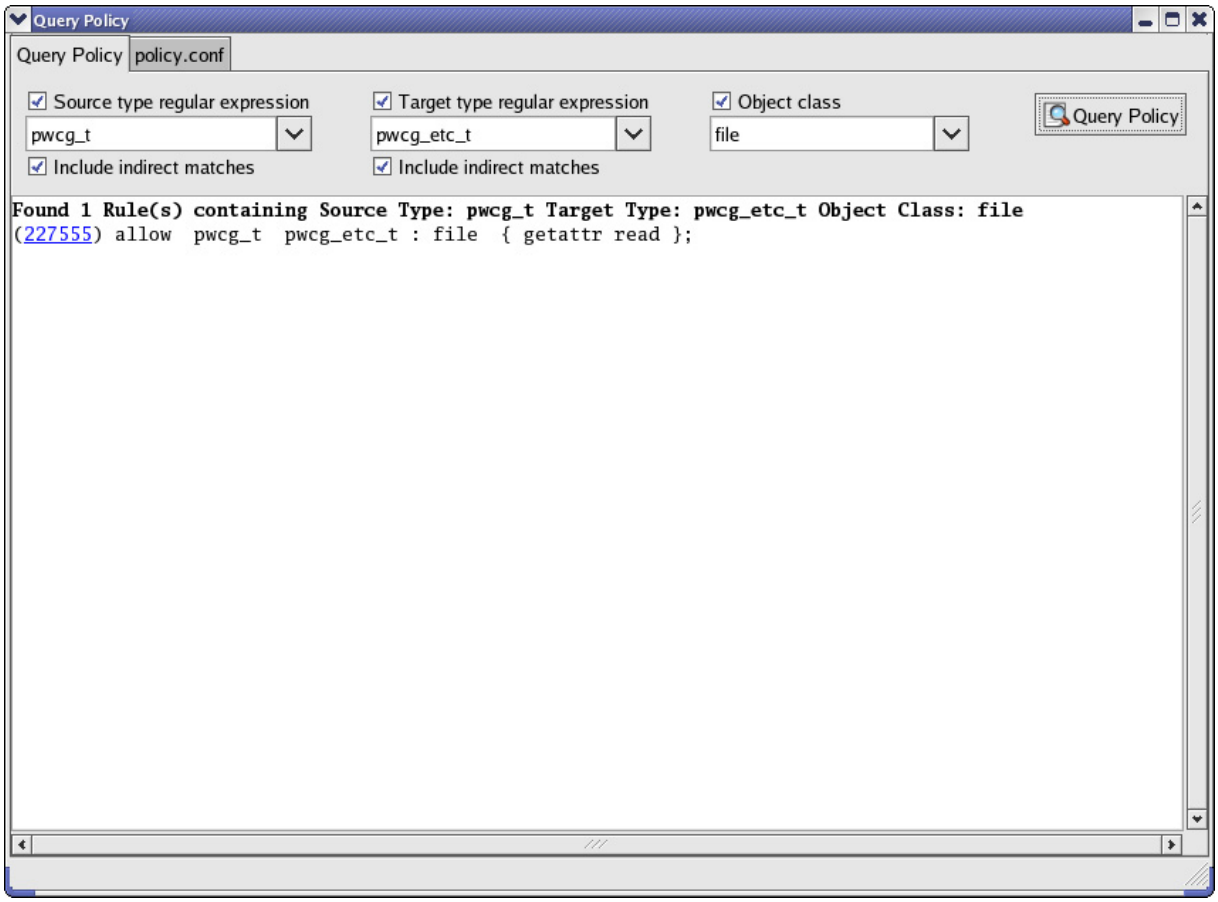


Figuur 20: Toevoegen van een filter



Figuur 21: Instellen van een filter

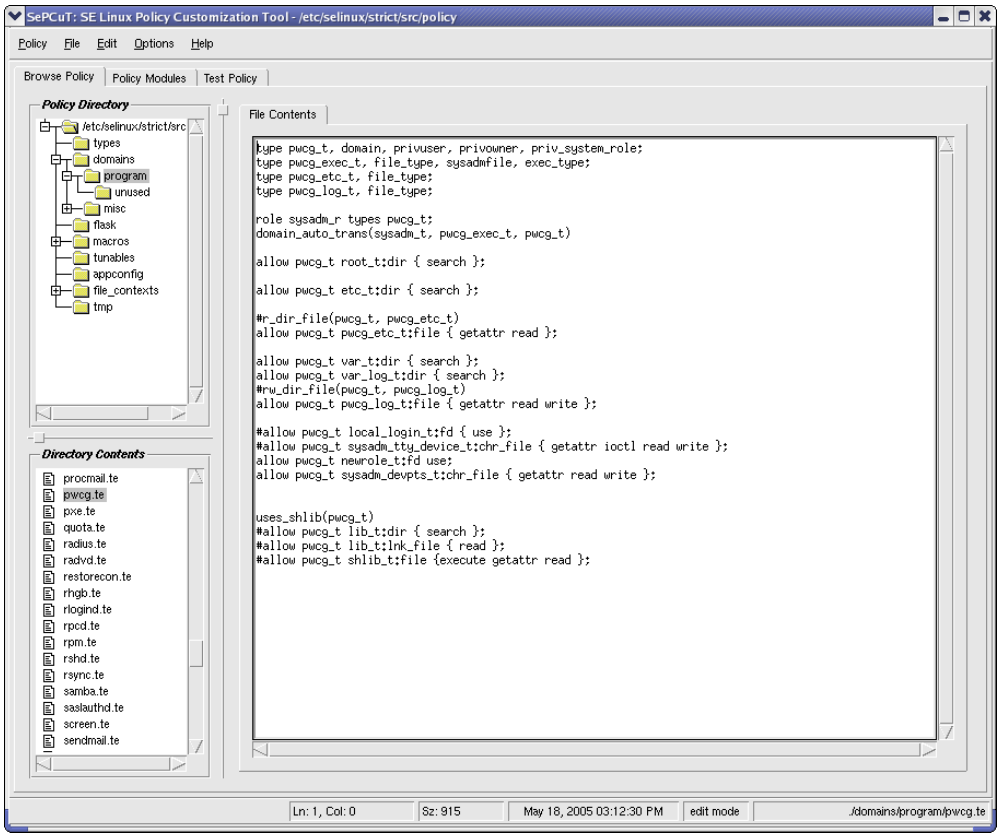
Seaudit heeft ook nog een andere functie. Het kan query's op de policy uitvoeren. In Figuur 22 hebben we naar regels gezocht waarvan het source type `pwcg_t` is, het target type `pwcg_etc_t` en de object class file is



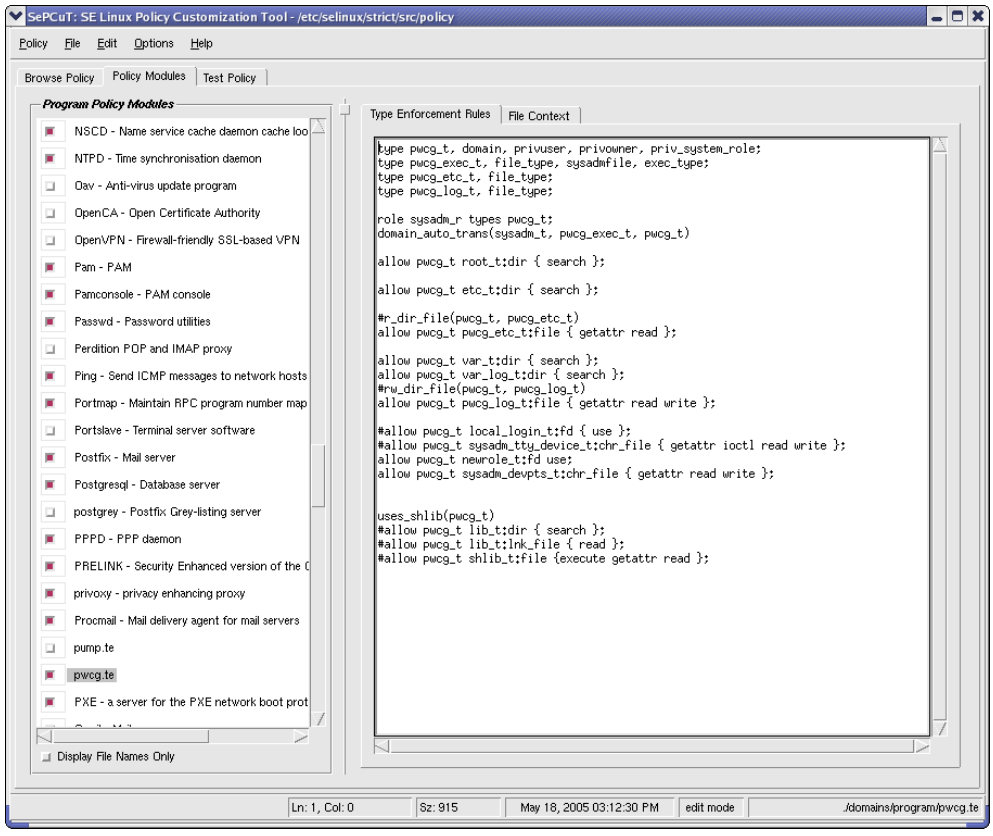
Figuur 22: Het doorzoeken van de policy

6.3 Sepcut

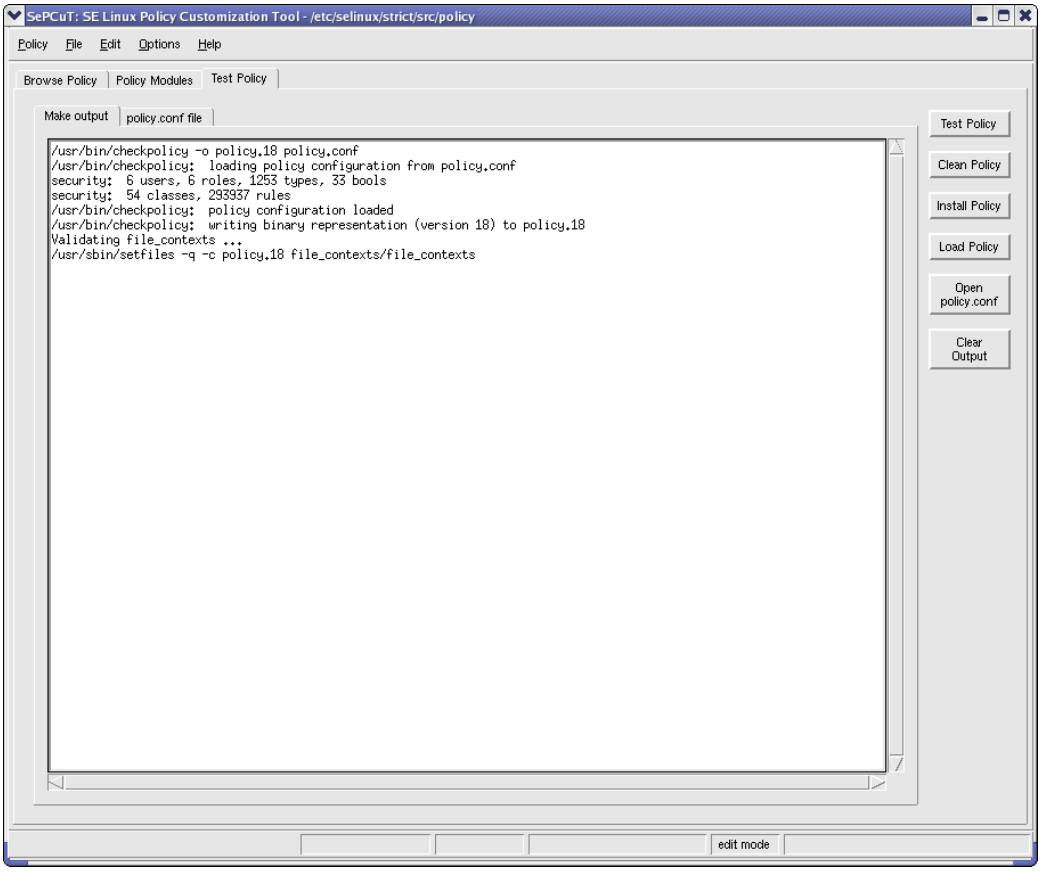
Dan hebben we nog een nuttig programma waarmee we de policy directory makkelijker kunnen doorbladeren en waarmee we de policy kunnen testen en laden. Deze tool heet sepcut. Het heeft ook weer verschillende tabbladen. Het eerste tabblad (Figuur 23) is om door de policy directory te bladeren. Het tweede tabblad (Figuur 24) geeft de verschillende programma's weer waarvoor een policy is geschreven. En met het derde tabblad (Figuur 25) kunnen we de policy testen en laden.



Figuur 23: Het doorzoeken van de policy directory



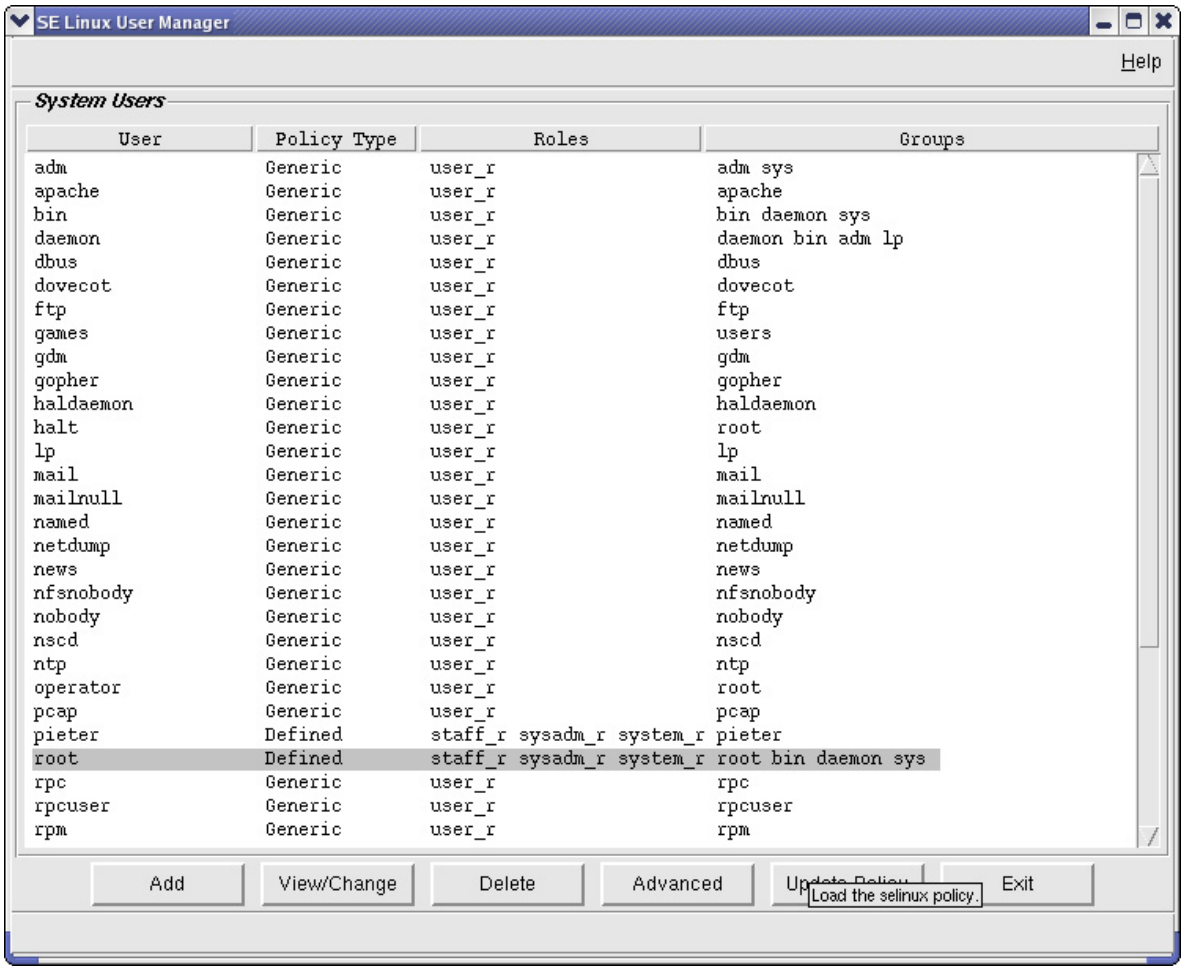
Figuur 24: De programma's waarvoor regels geschreven zijn



Figuur 25: Het testen en laden van de policy

6.4 Seuserx

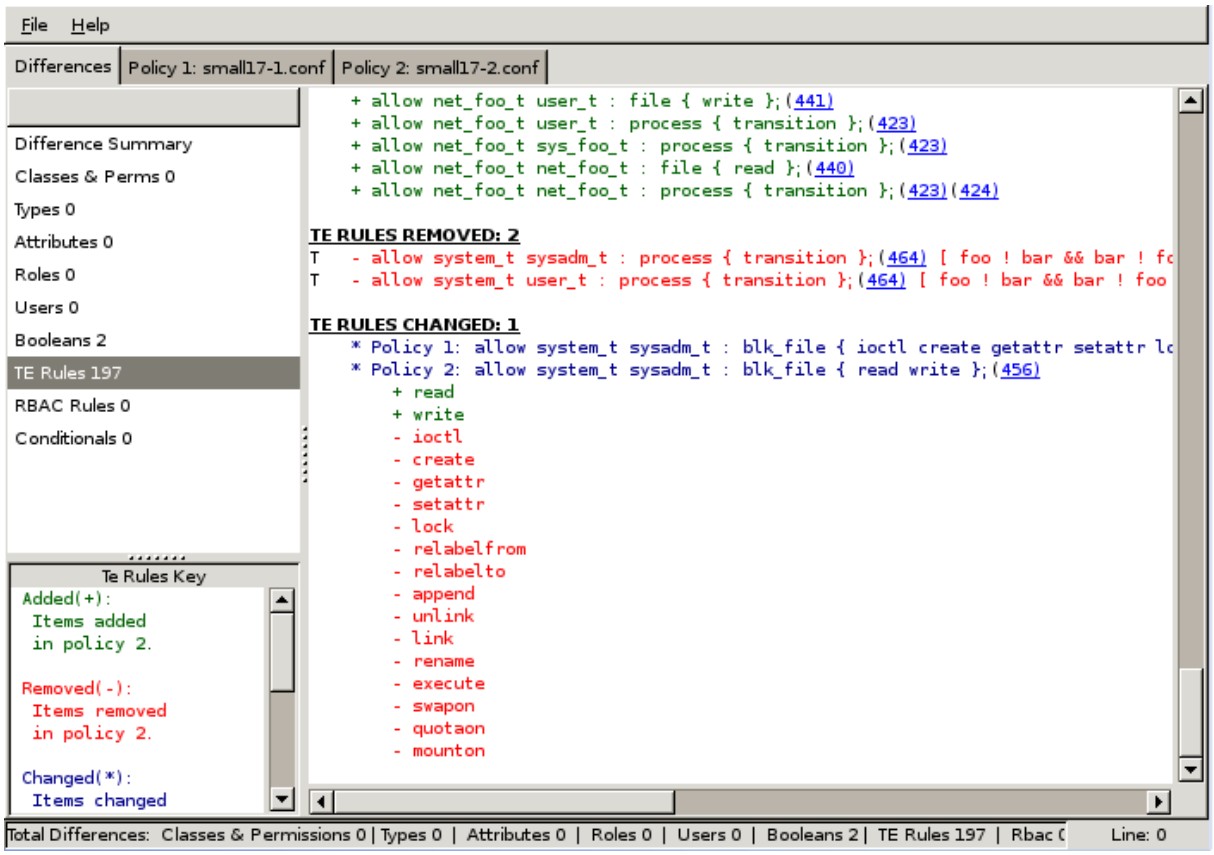
Deze tool maakt het mogelijk om gebruikers te beheren. Zoals men in Figuur 26 kan zien kunnen we een gebruiker toevoegen, verwijderen, …



Figuur 26: Het beheren van gebruikers

6.5 Sediff

Met deze tool kunnen we de verschillen tussen twee policys makkelijk vinden. Dit is zeer nuttig wanneer we een bestaande policy van een distributie naar een andere distributie willen omzetten. In Figuur 27 wordt een venster weergegeven waarin verschillen tussen 2 policys staan.



Figuur 27: De verschillen tussen twee policys

7 Beveiligen van eigen programma

Om de werking van SELinux nog wat beter te verduidelijken hebben we zelf een programma geschreven en dit beveiligd met SELinux. De meeste programma's die op Linux werken zijn redelijk complex om te beveiligen, denk maar aan ssh, httpd, enz..
Het programma dat wij geschreven hebben leest een file in `/etc` en schrijft de inhoud van deze file een aantal keer in de log file. Hoeveel keer het in de log file wordt geschreven is afhankelijk van de waarde die we ingeven tijdens de uitvoering van het programma. Hieronder vindt men de code van ons programma met de nodige uitleg.

```
#include <stdio.h>
// We hebben deze include nodig voor standaard input en ouput.
int main()
{
        int aantal,i;
        char *string[80];
        FILE *stream;
        stream = fopen("/etc/pwcg.conf","r");
//geeft error als we file niet kunnen openen.
        if (stream==NULL)
        {
          printf("kan file /etc/pwcg.conf niet openen.\nDeze
          file bevat een simpele string die we in een logfile
          gaan schrijven”);
        }
        else
        {
          fscanf(stream,"%s",string);
//we zetten de string uit de file in de variabele string
          printf("geef het aantal keren dat de string uit de
          file in de log moet komen en druk op enter:\n");
          scanf("%d", &aantal);
```

```
//we lezen het aantal keer in dat de string in de log file
//moet worden geschreven
          fclose(stream);
//we sluiten de huidige stream omdat we een nieuwe moeten
//openen voor de log file
          stream = fopen("/var/log/pwcg.log","w");
//we openen de log file om in te schrijven
          for(i=0;i<aantal;i++)
          {
               fprintf(stream,"%s\n",string);
//we schrijven een aantal keer de string in de log file
          }
          printf("%s is %d keer in /var/log/pwcg.log
          geschreven\n",string, aantal);
//een berichtje dat het gelukt is
          return 0;
//stop het programma
        }
}
```

Nu moeten we dit programma nog beveiligen met SELinux. Hiervoor moeten we een aantal stappen doorlopen.

1. kijken welke files er nodig zijn voor het programma
2. bepalen van de security context van deze files
3. kijken welke security context er nodig is voor het nieuwe domain
4. maken van de basis fc file
5. maken van de basis te file
6. toepassen van de nieuwe policy en file context
7. herhalen van deze stappen
    a. testen van het programma
    b. bekijken van de log files
    c. aanpassen van de te en fc files

7.1 Achterhalen welke files het programma gebruikt

Het eerste wat we moeten doen is achterhalen welke files het programma tijdens de uitvoering gebruikt. Het gaat hier om 3 files:

De eerste file is het programma zelf, in gecompileerde vorm.
De tweede file is de config file `/etc/pwcg.conf` waaruit we een string gaan lezen.
De derde file is de log file `/var/log/pwcg.log` waarin we de string uit de config file gaan schrijven.

7.2 Bepalen van de security context van deze files

We openen de `pwcg.conf` file met het commando:

```
Pico /etc/pwcg.conf
```

En zetten iets in deze file bijvoorbeeld “selinux”.

Het programma maakt de file `pwcg.log` aan. Maar deze kunnen we ook beter op voorhand aanmaken zodat we deze in de volgende stappen ook rechtstreeks de juiste security context kunnen geven.

```
Touch /var/log/pwcg.log
```

De files zijn aangemaakt en we kijken nu welke security context ze bezitten. Dit kunnen we met de volgende commando’s:

```
ls –Z /etc/pwcg.conf
```
geeft:
```
-rw-r- -r- -   root root system_u:object_r:etc_t  pwcg.conf
```

```
ls –Z /var/log/pwcg.log
```
geeft:
```
-rw-r- - - - - root root system_u:object_r:var_log_t pwcg.log
```

Dit zijn de standaard contexts die worden toegewezen indien er een nieuwe file in deze directories word gemaakt.

7.3 Bepalen van de security context voor het nieuwe domain

We gaan ons programma in een nieuw domain laten werken. Dit verhoogt de veiligheid van ons programma. Om ons programma in het juiste domain te laten werken moeten we de security context van een file aanpassen en kiezen in welk domain het programma uiteindelijk zal draaien. De meest evidente keuze is het programma te laten draaien in het `pwcg_t` domain. Om dit goed te laten verlopen moeten we de file contect van het programma nog aanpassen en deze maken we van het type `pwcg_exec_t`.

7.4 Maken van de basis fc file

We weten het domain al waarin we ons programma zullen laten draaien. Bijgevolg weten we ook al welke security context het programma moet hebben. Dus we kunnen de eerste regel van onze fc file al maken. Deze file moeten we natuurlijk wel op de juiste plaats aanmaken. Het pad voor deze file is `/etc/security/selinux/src/policy/file_contexts/program/pwcg.fc`.
De eerste regel in onze file wordt:

```
/sbin/pwcg -- system_u:object_r:pwcg_exec_t
```

De twee streepjes in het midden willen zeggen dat het een gewone standaard file is.

Deze regel zorgt voor de juiste file context van het programma.
We hebben ook nog twee andere files nodig bij ons programma namelijk de config file en de log file. Ook voor deze hebben we een juiste file context nodig. De volgende twee regels zorgen voor de juiste context van deze twee files.

```
/etc/pwcg.conf -- system_u:object_r:pwcg_etc_t
```

```
/var/log/pwcg.log -- system_u:object_r:pwcg_log_t
```

zodat de file `pwcg.te` er uiteindelijk als volgt uitziet:

```
#eigen programma pwcg
/sbin/pwcg -- system_u:object_r:pwcg_exec_t
/etc/pwcg.conf -- system_u:object_r:pwcg_etc_t
/var/log/pwcg.log -- system_u:object_r:pwcg_log_t
```

7.5 Maken van de basis te file

Nu gaan we beginnen aan het eigenlijke werk van de policy voor ons programma. Het eerste wat we in deze file moeten definieren zijn de types die we gaan gebruiken. We hebben er 4. Het eerste is voor het domain, de tweede is voor het programma, de derde is voor de config file en de vierde is voor de log file.
Hieronder volgen de 4 regels:

```
type pwcg_t, domain;
type pwcg_exec_t, file_type, sysadmfile, exec_type;
type pwcg_etc_t, file_type;
type pwcg_log_t, file_type;
```

De eerste regel spreekt voor zich. Het type `pwcg_t` is van het domain type. Het programma zal in dit domain worden uitgevoerd.
De tweede regel zegt dat het type `pwcg_exec_t` van het programma een file is waarop iemand uit de `sysadm_r` rol de volledige rechten op heeft en dat het een uitvoerbaar bestand is.
De derde en vierde regel bepalen de types van de config file en de log file. We kunnen hier ook de sysadmfile bijzetten. Dit zou ervoor zorgen dat personen in de `sysadm_r` rol alle rechten op deze file hebben. Maar wij willen dat niemand deze file kan lezen alleen het programma zelf, daarom hebben we er dat niet bijgezet. Normaal wordt dit wel gedaan zodat de system administrator steeds deze files kan aanpassen.

7.6 Toepassen van de nieuwe policy en file context

Nu gaan we onze nieuwe policy file toepassen en zorgen dat de files de juiste context hebben. Dit doen we door de volgende commando's achter elkaar uit te voeren.

```
Cd /etc/security/selinux/src/policy
Make load
Setfiles file_contexts/file_contexts /sbin/pwcg
Setfiles file_contexts/file_contexts /etc/pwcg.conf
Setfiles file_contexts/file_contexts /var/log/pwcg.log
```

Het belangrijkste is dat de files nu juist gelabeld zijn. We moeten ook al een basis te file aanmaken anders gaat die de fc file waarin de contexten staan gewoon negeren. Nu gaan we controleren of alle contexten juist zijn met:

```
Ls –Z /sbin/pwcg              system_u:object_r:pwcg_exec_t
Ls –Z /etc/pwcg.conf          system_u:object_r:pwcg_etc_t
Ls –Z /var/log/pwcg.log       system_u:object_r:pwcg_log_t
```

We zien dat de files de juiste context bezitten.
We kunnen nu ons programma al eens testen en kijken naar de log file. We moeten er wel zeker voor zorgen dat we nog in permissive mode zitten zodat alle errors in de log file komen. Bij enforcing stopt het programma wanneer de eerste error optreedt en krijgen we de rest van de errors niet te zien. De log file ziet er dan zo uit:

```
audit(1115927497.464:0): avc:  denied  { read } for  pid=654
exe=/sbin/pwcg name=pwcg.conf dev=hda1 ino=320147
scontext=root:sysadm_r:sysadm_t
tcontext=system_u:object_r:pwcg_etc_t tclass=file

audit(1115927497.467:0): avc:  denied  { getattr } for
pid=654 exe=/sbin/pwcg path=/etc/pwcg.conf dev=hda1 ino=320147
scontext=root:sysadm_r:sysadm_t
tcontext=system_u:object_r:pwcg_etc_t tclass=file
```

```
audit(1115927498.449:0): avc:  denied  { write } for  pid=654
exe=/sbin/pwcg name=pwcg.log dev=hda1 ino=132095
scontext=root:sysadm_r:sysadm_t
tcontext=system_u:object_r:pwcg_log_t tclass=file

audit(1115927498.451:0): avc:  denied  { getattr } for
pid=654 exe=/sbin/pwcg path=/var/log/pwcg.log dev=hda1
ino=132095 scontext=root:sysadm_r:sysadm_t
tcontext=system_u:object_r:pwcg_log_t tclass=file
```

We zien dus duidelijk dat er nog heel wat errors zijn. Maar deze errors zijn gelukkig niet zo moeilijk op te lossen. Deze hebben allemaal betrekking op het lezen van files. Maar we zien ook dat de `scontext root:sysadm_r:sysadm_t` is. Dit zou normaal niet mogen. We willen dat het progamma in het `pwcg_t` domain draait. En we willen ook dat dit domain de config file kan lezen en de log file kan schrijven.
Het eerste wat we nu moeten doen is zorgen dat het programma in zijn eigen domain werkt. Dit kunnen we doen door 2 regels aan de te file toe te voegen.

```
role sysadm_r types pwcg_t;
domain_auto_trans(sysadm_t, pwcg_exec_t, pwcg_t)
```

De eerste regel geeft `sysadm_r` toegang tot het `pwcg_t` domain
De tweede regel zorgt ervoor dat als `sysadm_r` het programma met type `pwcg_exec_t` uitvoert, er een automatische transitie naar het `pwcg_t` domain is. Nu zou ons eerste probleem al opgelost moeten zijn. Dit kunnen we testen door de policy opnieuw te compileren, te laden, het programma uit te voeren en te kijken of ons programma al in het juiste domain draait.
En als we het testen met `ps ax --context` dan zien we inderdaad dat ons programma in het juiste domain werkt.
Maar er zijn nog steeds errors. Errors die we wel konden verwachten, we kunnen namelijk het type `pwcg_etc_t` nog altijd niet lezen en het type `pwcg_log_t` nog altijd niet lezen of schrijven.
Dit gaan we nu ook oplossen. Hiervoor komt er wel al wat meer code kijken. Het eerste wat we moeten toelaten is dat het `pwcg_t` domain de root directory kan

doorzoeken. Hij kan er niks lezen maar hij kan deze wel doorzoeken om zo de `/etc` en de `/var` directory te vinden. Dit doen we met de volgende regel:

```
allow pwcg_t root_t:dir { search };
```

Nu kunnen we de root (/) directory doorzoeken. Maar aangezien de config file in de `/etc` directory staat moeten we ook deze directory kunnen doorzoeken.

```
allow pwcg_t etc_t:dir { search };
```

De volgende stap is deze file leesbaar maken voor het `pwcg_t` domain. Dit kunnen we op 2 manieren doen.

De eerste manier is `pwcg_t` dit toelaten via een macro.

```
r_dir_file(pwcg_t, pwcg_etc_t)
```

De tweede manier is dit toelaten door een allow regel te schrijven.

```
allow pwcg_t pwcg_etc_t:file { getattr read };
```

Zo nu kan `pwcg_t` onze config file al lezen. We laten `pwcg_t` natuurlijk niet toe om hierin te schrijven omdat het programma dit niet nodig heeft.

De volgende stap die we moeten doen is `pwcg_t` toelaten om in onze log file te schrijven. Dit is ongeveer hetzelfde als hierboven vernoemt. Eerst moeten we het toelaten om de `/var` en de `/var/log` directory te doorzoeken.

```
allow pwcg_t var_t:dir { search };
allow pwcg_t var_log_t:dir { search };
```

Nu moeten we `pwcg_t` nog toelaten om in de log file te schrijven. Dit kunnen we weer op twee manieren: met een macro of met een allow regel.

```
rw_dir_file(pwcg_t, pwcg_log_t)
```

```
allow pwcg_t pwcg_log_t:file { getattr read write };
```

Om te kijken hoe ver we staan gaan we onze policy nog eens opnieuw compileren en laden. We krijgen nog steeds veel foutmeldingen op het scherm. Nu gaan we een andere methode gebruiken om onze foutmeldingen weg te werken.
We gaan het programma `audit2allow` gebruiken.
`Audit2allow` kan je verschillende opties meegeven:

-d leest de input van dmesg
-v verbose mode (laat meer zien dan alleen de allow regels)
-l maakt de allow regels vanaf de laatste keer dat de policy opnieuw geladen werd
-i geeft de input file waar de avc errors instaan op
-o geeft een output file waar de allow regels komen te staan op

Wij gebruiken meestal de opties –d en –l en dan krijgen we deze output.

```
Allow pwcg_t ld_so_cache_t:file { getattr read };
Allow pwcg_t lib_t:dir { search };
Allow pwcg_t lib_t:lnk_file { read };
Allow pwcg_t local_login_t:fd { use };
Allow pwcg_t shlib_t:file { execute getattr read };
Allow pwcg_t sysadm_tty_device_t:chr_file { getattr iotcl read
write };
```

Er zijn een aantal regels die betrekking hebben op libraries. Een programma dat gecompileerd wordt gebruikt veel bibliotheken. Dus het programma moet deze kunnen lezen en uitvoeren. Dit doen we met deze drie regels.

```
Allow pwcg_t lib_t:dir { search };
Allow pwcg_t lib_t:lnk_file { read };
Allow pwcg_t shlib_t:file { execute getattr read };
```

We kunnen dit ook op een andere manier doen. Dit is weer met een macro.

```
uses_shlib(pwcg_t)
```

Ons programma werkt nog niet. We zullen er eerst nog twee regels aan moeten toevoegen.

```
Allow pwcg_t local_login_t:fd { use };
Allow pwcg_t sysadm_tty_device_t:chr_file { getattr iotcl read
write };
```

De eerste regel laat `pwcg_t` toe om een file descriptor te gebruiken die gecrëerd is door het `local_login_t` domain. Deze file descriptor laat ons programma toe om een file te lezen of te schrijven

De tweede regel laat ons programma toe om op de terminal te schrijven en ervan te lezen. We schrijven ook een tekst op het scherm en we lezen het aantal keer dat de inhoud van de config file in de log file moet komen.

De uiteindelijke file ziet er zo uit:

```
type pwcg_t, domain;
type pwcg_exec_t, file_type, sysadmfile, exec_type;
type pwcg_etc_t, file_type;
type pwcg_log_t, file_type;

role sysadm_r types pwcg_t;
domain_auto_trans(sysadm_t, pwcg_exec_t, pwcg_t)

allow pwcg_t root_t:dir { search };

allow pwcg_t etc_t:dir { search };

#r_dir_file(pwcg_t, pwcg_etc_t)
allow pwcg_t pwcg_etc_t:file { getattr read };

allow pwcg_t var_t:dir { search };
allow pwcg_t var_log_t:dir { search };
```

```
#rw_dir_file(pwcg_t, pwcg_log_t)
allow pwcg_t pwcg_log_t:file { getattr read write };

allow pwcg_t local_login_t:fd { use };
allow pwcg_t sysadm_tty_device_t:chr_file { getattr ioctl read
write };


uses_shlib(pwcg_t)
#allow pwcg_t lib_t:dir { search };
#allow pwcg_t lib_t:lnk_file { read };
#allow pwcg_t shlib_t:file {execute getattr read };
```

Voor zo een klein programma te schrijven is er toch redelijk wat code nodig. SELinux beveiligingen schrijven voor grotere programma's is dus zeker niet simpel. Men moet eerst de werking van het programma volledig begrijpen. Zonder het programma te begrijpen is het schrijven van juiste regels niet haalbaar. We denken dus ook dat de ontwikkelaars van programma's zelf gaan zorgen voor de policy regels.

8 Conclusies

In ons eindwerk hebben we SELinux bestudeerd, waarmee een systeem heel uitgebreid beveiligd kan worden. Door middel van SELinux kan men de toelatingen van programma's en gebruikers controleren en beperken. De belangrijkste regel is hier dat een programma of gebruiker het minimum van rechten krijgt.

Het belangrijkste deel van SELinux is de policy. Een goede policy is de basis voor een veilig systeem. Fedora en Gentoo hebben de beste policy op dit moment, maar ze zijn beide nog niet ver genoeg ontwikkeld om een server volledig te beveiligen. De meeste basisprogramma's zoals `ssh, httpd, dhcpd, named, nscd, ntpd, portmap, snmpd, squid` en `syslogd` zijn al beveiligd. De uitgebreidere programma's zoals `mail server`, `tomcat`, `cocoon` en `nfs4` worden echter nog niet beveiligd door de policy. Bij Fedora en Gentoo wordt de policy wekelijks vernieuwd, maar bij Slackware is dit niet het geval. Slackware is een minder bekende distributie, dus zijn er ook minder mensen mee bezig. De policy is er dus nog niet optimaal en men zal deze ook nog verder moeten ontwikkelen.

SELinux is een zeer complexe vorm van beveiligen. Omdat men zoveel acties kan controleren moeten er veel regels geschreven worden. Om deze regels te schrijven is niet alleen een zeer goede kennis van SELinux nodig, maar moet men ook perfect weten hoe de verschillende daemons van Linux werken. Deze kennis van SELinux hebben we verworven, maar de tijd was te kort om alle daemons te begrijpen en te beveiligen. Om dezelfde reden is het voor gewone gebruikers te ingewikkeld om zelf policys te schrijven.

Verder is SELinux niet alleen een zeer complexe manier van beveiligen, maar ook zeer strikte. Strikt wil zeggen dat wat we in de policy niet toelaten, ook echt niet mogelijk is. Aangezien de beveiliging van computers en servers aan belang wint, is SELinux zeker een goede manier om voor deze beveiliging te zorgen.

Bovendien denken we dat SELinux de standaard manier van beveiligen gaat worden. Hiervoor moeten wel nog wat ontwikkelingen doorgevoerd worden. De gebruiker zal in de toekomst de werking van SELinux en alle daemons niet meer moeten kennen, omdat de ontwikkelaars van de daemons of van de verschillende distributies

waarschijnlijk zelf een bijbehorende policy zullen schrijven. Ook het aanpassen van de policy zal niet meer gebeuren op codeniveau. De kans is groot dat er grafische programma's gemaakt zullen worden die door een simpele muisklik de policy kunnen aanpassen. Fedora Core 3 heeft al enkele van deze programma's ontwikkeld.

Ondanks de weinige informatie die voorhanden was, hebben we toch meer geleerd over de werking van SELinux en de verschillende daemons. Bovendien hebben we een beter inzicht gekregen in het schrijven van policys.

9 Bibliografie

COAR, K., *Mandatory Versus Discretionary Access Control*, online, http://www.linuxplanet.com/linuxplanet/tutorials/1527/2/, 18 mei 2005.

COKER, F., *Security Enhanced Linux Implementation Lab* ,online, http://www.coker.com.au/selinux/talks/ibmtu-2004/linux20.doc, 17 mei 2005.

COKER, F., *Getting started with SELinux HOWTO: the new SELinux*, online, http://www.lurking-grue.org/GettingStartedWithNewSELinuxHOWTO.pdf, 18 mei 2005.

COKER, F., *Writing SE Linux policy HOWTO*, online, http://www.lurking-grue.org/WritingSELinuxPolicyHOWTO.pdf, 18 mei 2005.

COKER, R., *Introduction to SE Linux lecture notes*, online, http://www.coker.com.au/selinux/talks/ibmtu-2004/LINUX20.pdf, 18 mei 2005.

COKER, R., *SE Linux policy writing lecture notes*, online, http://www.coker.com.au/selinux/talks/ibmtu-2004/LINUX21.pdf, 18 mei 2005.

Gentoo x86 SELinux Handbook, online, http://www.gentoo.org/proj/en/hardened/selinux/selinux-x86-handbook.xml, 18 mei 2005.

LOEB, L., *Uncovering the secrets of SE Linux*, online, http://www-106.ibm.com/developerworks/library/s-selinux/index.html, 18 mei 2005.

Mccarty, B, *SELinux:NSA's Open Source Security Enhanced Linux*, Sebastopol, 2005.

SMALLEY, S., *Configuring the SELinux Policy*, online, http://www.nsa.gov/selinux/papers/policy2-abs.cfm, 17 mei 2005.

SMALLEY, S., FRASER, T., A *Security Policy Configuration for the Security-Enhanced Linux*, online, http://www.nsa.gov/selinux/papers/policy-abs.cfm,17 mei 2005.

WOOD, T., *Installing Security Enhanced Linux and Pluggable Authentication Modules on Slackware 9*, online, http://www.diyab.net/selinux/, 17 mei 2005.

10 Bijlages

10.1 NSA/CSS

NSA/CSS is een Amerikaans bedrijf dat zich bezighoudt met het coördineren en uitvoeren van gespecialiseerde activiteiten om Amerikaanse informatiesystemen te beschermen en om belangrijke informatie (“intelligence information”) te zoeken. Ze zijn altijd bezig met de laatste nieuwe ontwikkelingen op het gebied van datacommunicatie. NSA/CSS bestaat uit twee onderdelen: Signals intelligence (SIGINT), Information Assurance (IA).

SIGINT zorgt voor de belangrijke informatie die men kan gebruiken om vijanden van de Verenigde Staten altijd een stap voor te zijn. De geschiedenis van SIGINT loopt terug tot de tweede wereldoorlog, waar ze de militaire code van de Japanners gebroken hadden en zo een aantal plannen van de Japanners verijdeld hadden.

Het onderdeel IA beveiligt de Amerikaanse informatiesystemen. Omdat de wereld meer en meer op technologie georiënteerd wordt, is het belangrijk dat de geclassificeerde en gevoelige informatie die door de Amerikaanse overheidsapparatuur opgeslagen en verstuurd wordt, niet indringbaar is en blijft.

NSA/CSS is ook de grootste werkgever van wiskundigen in de Verenigde Staten. Een groot deel van deze wiskundigen wordt opgeleid in een school van de NSA/CSS. Buiten de opleiding tot wiskundige kan men er ook nog andere bachelor- en masteropleidingen volgen.

10.2 GNU

Het GNU-project werd in 1984 door Richard Stallman gelanceerd. De bedoeling was om een nieuw vrij Unix-achtig besturingssysteem voor computers te maken.

GNU is nauw verbonden met de Free Software Foundation, die dit als het centrale project beschouwt van de beweging voor vrije software. Zo is vanuit GNU de GPL-licentie (GNU General Public License) voor vrije software ontwikkeld.

De GNU-licentie betekent dat je de software gratis mag gebruiken, niet alleen privé maar ook commercieel. Vaak komt deze gratis software echter zonder garantie. Als er zich een probleem voordoet, kan je dat meestal melden aan de ontwikkelaar zodat deze het kan oplossen. Als je kunt programmeren, kan je door de openbaarheid van de broncode van de software zelf het probleem oplossen of zelfs een betere code schrijven. Zo help je mee aan het project. Des te groter en bekender een programma is, des te meer gebruikers en ontwikkelaars er zullen zijn. Het ontbreken van officiële support en garantie is daarom vaak geen probleem. De grote groep van ontwikkelaars en gebruikers ontdekt problemen en fouten snel. Zo kunnen ze ook snel worden opgelost.

De vrijheid van GNU/Linux bestaat uit vier grote groepen:

- Het programma gebruiken.
- Het programma bestuderen en aanpassen (broncode is vrij beschikbaar).
- Het aangepaste programma weer verspreiden en zo andere mensen helpen.
- De code van het aangepaste programma verspreiden zodat ook andere mensen deze code weer kunnen aanpassen en verspreiden.

Omdat alles vrij beschikbaar is, is er een zeer grote groep personen die aan de verdere ontwikkeling van Linux meewerkt.. Dit wordt algemeen beschouwd als de grote sterkte van Linux.

10.3 XML

Extensible Markup Language (XML) is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst. Deze representatie is zowel machineleesbaar als leesbaar voor de mens.

Met andere woorden: is een bepaalde manier om gegevens gestructureerd vast te leggen (bijvoorbeeld in een bestand, maar ook voor het doorsturen van informatie over het internet). Deze manier is gedefinieerd en mag iedereen gebruiken. XML is ontworpen om zowel door een programma als door een mens leesbaar te zijn.

XML is een vereenvoudigde vorm van SGML, Standard Generalized Markup Language, een heel complexe standaard die gebruikt werd om ingewikkelde documenten vorm te geven.
Een slechtere opvolger van SGML is HTML HyperText Markup Language. HTML heeft voor een doorbraak in SGML-achtig vormgegeven tekst gezorgd, maar gegevens die op een HTML-pagina staan zijn voor computers niet als zodanig te herkennen.

XML zorgt nu juist voor die herkenbaarheid van gegevens. Voorbeeld: een XML-bestand dat een muziek-playlist beschrijft zou er als volgt uit kunnen zien:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<playlist name="mylist">
 <song>
  <title>Little Fluffy Clouds</title>
  <artist>the Orb</artist>
 </song>
 <song>
  <title>Goodbye mother Earth</title>
  <artist>Underworld</artist>
 </song>
</playlist>
```

Het gaat in dit bestandsformaat dus meer om de structuur van informatie, dit in tegenstelling tot HTML, waarin het meer gaat om de presentatie van de informatie. In een HTML-bestand beschrijven de tags wel hoe informatie moet worden gepresenteerd maar niet wat deze informatie betekent.

De afspraken over de te gebruiken tags in de "standaard" dialecten worden formeel vastgelegd in zogenaamde DTD's (Document Type Definitie) of in de nieuwere XML Schema Definities (XSD). Naast de te gebruiken tags wordt hierin ook beschreven welke gegevens acceptabel zijn en hoe ze precies moeten worden opgegeven (bijvoorbeeld postcode bestaat uit 4 cijfers, 1 spatie, 2 letters). Het verschil tussen DTD en XSD is dat XSD schema's hierin meer uitdrukkingskracht hebben; daarnaast is XSD zelf ook een XML-dialect dat met alle XML-tools kan worden bewerkt.

Hoe de gegevens opgemaakt zullen worden, geef je op met een XSL document Extensible Stylesheet Language. Het is ook enigszins mogelijk om een XML-document op te maken met een CSS-document. CSS is echter beter geschikt voor XHTML. Op een dergelijk manier geef je in XML middels XSD en XSL een keurige scheiding tussen opmaak en inhoud. Ook XSL is zelf een (standaard) XML-dialect.

Data in XML-formaat kunnen door middel van XSLT-transformaties worden omgezet naar andere formaten zoals HTML, WML of PDF. In het geval van de transformatie naar HTML kan deze bewerking zowel in de browser (op het moment van tonen) als op voorhand gebeuren.

Hoewel de XML-tags in principe vrij te kiezen zijn, is het bij uitwisseling van gegevens wel handig als er een gemeenschappelijke standaard wordt afgesproken. Op deze manier ontstaan er allerlei XML-dialecten, elk met een eigen specifieke toepassing. Een voorbeeld van een "standaard" XML-dialect is de zogenaamde RSS-standaard (Rich Site Summary of Really Simple Syndication) waarmee nieuwssites hun headlines kunnen uitwisselen. Van nieuwssites zoals NU nl. en SlashDot zijn bijvoorbeeld zogenaamde RSS-feeds beschikbaar.

10.4 SSH

SSH staat voor Secure Shell. Het is een protocol uit de applicatielaag van de TCP/IP protocol groep dat over TCP draait, normaal over TCP poort 22. De term SSH werd gemakshalve ook gebruikt voor het client-programma dat het protocol toepast. (Het server-programma heet dan weer sshd, 'Secure Shell daemon'.) SSH vervangt oudere protocols zoals telnet, rlogin, en rsh/rexec door een beveiligbare variant daarvan.

SSH maakt het mogelijk om op een geëncrypteerde manier in te loggen op een andere computer, en op afstand commando's op de andere computer uit te voeren via een shell. Omdat SSH met encryptie werkt, is het voor afluisteraars zo goed als onmogelijk om wachtwoorden of commando's te achterhalen, ook al wordt de (internet)connectie afgetapt.

SSH heeft ook de mogelijkheid om X11-connecties en TCP/IP-poorten door te sturen (forwarding). Het forwarden van X11 maakt het mogelijk om met encryptie te werken in een grafisch programma dat met X11 werkt, terwijl het programma niet draait op de computer waar de gebruiker achter zit, maar op de andere computer (vanwaar X11 geforward wordt).

Er bestaan ssh-programma's voor een groot aantal besturingssystemen. De bekendste versie is OpenSSH, een Open Source implementatie die door de programmeurs van het OpenBSD project ontwikkeld wordt.