

# Haking

NR 01 (lipiec-sierpień) CENA 29,80 ZŁ STAWKA VAT 0% NAKŁAD 6000 EGZ.

## Oracle

z punktu widzenia intruza

Jak (i po co)  
zainstalować  
**honeypota**

Przegląd bibliotek  
kryptograficznych  
na licencji GPL

Przechwytywanie  
połączenia danych  
w protokole **FTP**

**387** plakat z numerami portów  
+ opisy na CD  
**trojanów**



# SELinux – w służbie bezpieczeństwa

Grzegorz B. Prokopski

**D**la systemu Linux w ostatnich latach powstało wiele oprogramowania, którego zadaniem jest podnoszenie bezpieczeństwa systemu. Czy zatem istnieje potrzeba tworzenia jeszcze jednego, kolejnego rozwiązania? Na czym polega odmienność i siła SELinuxa? Skąd pochodzi, kto za nim stoi i na jakich zasadach jest dostępny? Jak działa, kto może i kto powinien go używać?

Na te i na inne pytania postaram się odpowiedzieć w poniższym artykule o dość rewolucyjnej, choć czerpiącej ze standardów, naturze Security Enhanced Linuxa, czyli Linuksa o wzbogaconym bezpieczeństwie. Zapraszam do lektury!

## Czym jest i skąd pochodzi

Najkrócej mówiąc *SELinux* jest systemem z *MAC* (ang. *Mandatory Access Control*), czyli *Obowiązkową kontrolą dostępu*, który realizuje politykę *RBAC* (ang. *Role Based Access Control*), czyli *Kontroli dostępu opartej na rolach* za pomocą *DTAC* (ang. *Dynamically Typed Access Control*) tłumaczonego również jako *Domenowy system kontroli dostępu*. Definicja wydaje się dość skomplikowana i jej dokładniejszym wyjaśnieniem zajmę się w dalszej części artykułu.

Natomiast znacznie bardziej zrozumiała jest historia SELinuxa. Prace nad nim są sponsorowane przez amerykańską *NSA* (ang. *National Security Agency*) a pieniądze te trafiają do pracującej nad projektem SELinuxa – *Secure Computing Corp.*, która to firma jest również właścicielem wykorzystywanych w nim patentów. Jednak idea i implementacja, których wynik znamy dziś pod nazwą SELinux, nie są wcale młode.

Od 1992 roku w ramach projektu *Distributed Trusted Mach (DTMach)* rozwijano rozwiązania, które następnie zostały przejęte przez system operacyjny *Fluke*, dla którego w ramach projektu *Flux* rozwijano architekturę *Flask*. Jakkolwiek brzmi to skomplikowanie, to prawdopodobnie najistotniejszym faktem jest, że to właśnie architekturę *Flask* zintegrowano z jądrem Linuksa, a powstały projekt ochrzczono imieniem *SELinux*.

## Co pod maską

Można wymienić trzy główne elementy składowe SELinuxa:

### Jądro

SELinux używa obecnie (bo nie zawsze tak było) infrastruktury jądra Linuksa znanej pod nazwą *LSM* (ang. *Linux Security Modules - Moduły bezpieczeństwa Linuksa*). Dostarcza ona interfejsów pozwalających kontrolować dostęp do obiektów systemu podczas wykonywania działań przez użytkowników (otwarcie pliku, utworzenie katalogu, bindowanie do portu itd.). SELinux podłącza się do tego interfejsu i wymusza w systemie własną politykę bezpieczeństwa. Natomiast z punktu widzenia administratora instalującego SELinuxa - stanowi on po prostu łatę na jądro.

### Zmodyfikowane kluczowe programy

SELinux w większości przypadków pozwala, aby programy działające w systemie nie musiały rozumieć, że system ten nie jest zwykłym Linuksem. Jednak pewne kluczowe dla bezpieczeństwa i w ogóle działania systemu programy muszą zostać rozszerzone o obsługę SELinuxa. Do tych programów należą między innymi: *ssh*, *te*, *ps*, *xm*, *login*... Administrator systemu musi podczas instalacji albo pobrać zmodyfikowane już wersje używanych programów, albo zaaplikować odpowiednie łatę na źródła używanych programów.

### Zasady, czyli policy

Określają prawa dostępu, prawa wykonywania działań w systemie, zachowanie się systemu. Jest to w praktyce najważniejsza część systemu SELinux, gdyż to właśnie te zasady decydują o skutecznym działaniu całego systemu. W niniejszym artykule poświęcę część miejsca na wyjaśnienie jak konstruowane są elementy *policy*.

Administrator systemu generalnie nie powinien być zmuszony do pisania własnych zasad w *policy* (chyba, że istnieją pewne niestandardowe wymagania co do kształtu polityki bezpieczeństwa), ale powinien umieć je czasem zmodyfikować do swoich potrzeb, a na pewno powinien rozumieć zasady już istniejące.

## O MAC (i ACL) słów kilka

Ponieważ SELinux jest systemem realizującym *MAC*, to warto wiedzieć co to dokładnie oznacza. Główna zasada w systemach z *MAC* brzmi: *użytkownik NIE decyduje o zabezpieczeniach i prawach dostępu do obiektów*. Prawa i zabezpieczenia o których mowa są definiowane ogólnie, zgodnie z *polityką bezpieczeństwa* i w przypadku SELinuxa znajdują swoje odbicie w zapisach *policy*. Ponieważ SELinux bywa porównywany z systemami z *ACL*, to warto wiedzieć, że:

- *ACL* nie realizują *MAC*,
- *ACL* są w sumie bardziej skomplikowane i pracochłonne w konfiguracji i kontroli poprawności.

Można też powiedzieć, że *ACL* zajmują się *jednostkowymi przypadkami*, natomiast jak się wkrótce okaże - SELinux realizuje *MAC* przez określanie *ogólnych* zasad. *ACL*-e nie znalazły się w jądrach z serii 2.4 także dlatego, że deweloperzy jądra nie uważają ich za dobry sposób kontrolowania uprawnień użytkowników w systemie. W praktyce znajdują one zastosowanie najczęściej w połączeniu z serwerem *Samba*, gdyż pozwalają wtedy ustalać uprawnienia do plików z poziomu klientów windowsowych.

## Co to jest DTAC

W języku angielskim oznacza to *Dynamically Typed Access Control*, co bywa tłumaczone również jako *domenowy system kontroli dostępu*. Najważniejsze cechy *DTAC* to:

- każdy obiekt (np. plik, port, urządzenie sieciowe) posiada pojedynczy typ,
- typ obiektu jest wymuszany ogólnymi regułami (nie zależy od decyzji użytkownika, ale administratora komponującego *policy*),
- w systemie można tworzyć niemal dowolną ilość reguł (rzędu 100 tysięcy) - system jest na to przygotowany,
- do danego typu tworzone są zestawy reguł decydujące o zachowaniu się systemu przy wykonywaniu akcji na obiektach danego typu (więcej na ten temat będzie za chwilę)
- kontrola i testowanie poprawności reguł jest łatwiejsze - IBM opracowuje narzędzie, które pozwala na automatyczną kontrolę spójności i poprawności polityki bezpieczeństwa *DTAC*.

## Co to jest RBAC

W języku angielskim oznacza to *Role Based Access Control* czyli *System kontroli dostępu bazujący na rolach*. Chociaż pojęcie to nie jest zbyt często używane, to warto wiedzieć, że np. standardowy system uniksowy jest systemem z *RBAC*. I rzeczywiście, gdy

przyjrzymy się bliżej - mamy tzw. *użytkowników systemowych*, z których każdy ma prawo wykonywać pewien zakres działań (tzn. realizować pewną rolę w systemie).

Drugim elementem kwalifikującym standardowy system uniksowy do systemów z *RBAC* jest Istnienie *SUID*-ów, za pomocą których można zmienić rolę, w ramach której dane działanie jest wykonywane. A jak to się ma do SELinuxa?

- SELinux jest systemem z *RBAC*, którego zasady są realizowane za pomocą *DTAC*,
- SELinux rozszerza filozofię *SUID*ów,
- SELinux rozszerza rozumienie ról użytkowników.

Ogólnie rzecz biorąc - SELinux pozwala w sposób znacznie bardziej elastyczny, dokładniejszy i bardziej dostosowany do konkretnych potrzeb określać i realizować role w ramach systemu.

## SELinux – system w systemie

Znając już podstawowe pojęcia - spróbujmy lepiej określić w jaki sposób działa SELinux.

- jeden użytkownik w systemie może posiadać wiele ról, przy czym jedna z nich jest rolą domyślną,
- każda rola użytkownika posiada zestaw domen dostępu, a jedna domena jest domyślną dla danej roli; w danej chwili program działający jako dany użytkownik może wykonywać jedną rolę w jednej domenie, mogą następować przełączenia między rolami i domenami,
- dla wybranych skrzyżowań domeny i każdego typu definiowane są reguły zachowania się (prawa i inne) - tak więc efektywne uprawnienia procesu w systemie zależą od domeny, w której w danej chwili działa,
- SELinux potrafi też współpracować ze standardowym mechanizmem *Capabilities* umożliwiając ustalanie zestawu *Capabilities* startującego programu zależnie od domeny procesu, który próbuje go uruchomić.

Trzeba jeszcze rozumieć jedną istotną rzecz: w jaki sposób reguły SELinuxa współgrają ze standardowym modelem uprawnień *UGO* (ang. *User, Group, Others* - *Użytkownik, Grupa, Inni*) systemu uniksowego. Generalną zasadą jest, że SELinux ogranicza uprawnienia do wykonywania zadań w stosunku do systemu bez SELinuxa, natomiast *nie nadaje dodatkowych uprawnień*. których użytkownik nie miał w normalnym systemie bez SELinuxa.

Z praktycznego punktu widzenia jest to podejście bardzo poprawne, gdyż w przypadku zaistnienia błędów w *polityce bezpieczeństwa* SELinuxa w najgorszym wypadku, gdyby nawet *policy* SELinuxa pozwalało wszystkim na wszystko, otrzymalibyśmy zwykły system uniksowy, którego zabezpieczenia wcale nie są takie złe (przecież używamy ich na co dzień!).

## Security Context procesu i obiektu

Konstrukcję SC, czyli *kontekstu bezpieczeństwa* procesu, najlepiej wyjaśnić na przykładzie. Po zalogowaniu się do systemu z SELinuxem przez użytkownika *greg* jego proces powłoki (ang. *shell*) będzie działał z następującym kontekstem bezpieczeństwa:

```
greg:user_r:user_t
```

gdzie:

- *greg* - identyfikator użytkownika uniksowego,
- *user\_r* - rola domyślna użytkowników po zalogowaniu się,
- *user\_t* - domena domyślna roli *user\_r*.

Tak naprawdę to obiekty w systemie także posiadają pełny SC, np. można zdefiniować SC portu TCP jako

```
system_u:object_r:http_port_t
```

Przy tym znacząca jest jedynie część ostatnia: *http\_port\_t*, która w przypadku obiektów nazywana jest *typem*, a nie *domeną*, jak to ma miejsce w przypadku procesów. Należy też mieć jasność co do tego, że z danej roli może korzystać (zgodnie z zapisami w *policy*) wielu użytkowników uniksowych – podobnie, jak do jednej domeny może mieć dostęp wiele ról.

## Główne rodzaje reguł SELinuxa

Reguły SELinuxa są definiowane dla par *domena* i *typ*. Można wyróżnić najczęściej używane rodzaje reguł:

- definiowanie operacji dozwolonych, jak czytanie, pisanie, tworzenie, wykonywanie, bindowanie (do portu) itp,
- dozwolona zmiana roli w ramach użytkownika (i w ogóle lista ról dozwolonych dla użytkownika),
- dozwolona zmiana domeny w ramach roli (i w ogóle lista domen dozwolonych dla roli),
- automatyczna (wymuszona, jak przy SUID-zie) zmiana roli bądź domeny przy wykonywaniu programu, lub przy wykonywaniu operacji na obiekcie (np. automatyczna zmiana typu przy tworzeniu pliku).

W chwili obecnej w standardowej *policy* zlikwidowano wszystkie wymuszenia zmiany roli, zastępując je wymuszeniami zmiany domeny.

## Przykład klienta IRC i przeglądarki internetowej

Patrząc na historię błędów w klientach IRC pozwalających przejąć atakującemu kontrolę nad maszyną, na której uruchomiono

klienta (przynajmniej w ramach uprawnień użytkownika) nie powinien dziwić fakt, że w standardowej *policy* znajdują się zapisy traktujące te programy w sposób specjalny:

- wprowadzono domenę *user\_irc\_t*,
- przy wykonaniu programu klienta IRC (którego plik wykonywalny także posiada specjalny typ: *irc\_exec\_:* następuje wymuszona zmiana domeny w *Security Context* procesu (*user:user\_r:user\_t => user:user\_r. user\_irc\_t*),
- proces działający w domenie *user\_irc\_t* ma bardzo mocno ograniczone prawa - praktycznie rzecz biorąc ma możliwość pisania tylko do logów i własnych plików konfiguracyjnych,
- należy zauważyć, że użytkownik i rola nie podlegają modyfikacji (to nie jest SUID).

Podobne niebezpieczeństwa niesie ze sobą używanie przeglądarek internetowych. Są to zwykle duże i skomplikowane programy, co niestety przekłada się na ilość błędów i wynikających z nich zagrożeń. Dlatego dla nich w sposób analogiczny:

- wprowadzono typ plików wykonywalnych *netscape\_exec\_t*.
- podczas *exec* o pliku o powyższym typie następuje przełączenie procesu do domeny *user\_netscape\_:* która ma ograniczone prawa:
- może tworzyć pliki o typie *user\_netscape\_t* i *user\_netscape\_rw\_t* (ten typ służy także do oznaczania plików konfiguracyjnych),
- może wykonywać pobrane programy, ale tylko w ramach bieżącej domeny,
- może drukować (prawo wykonywania programu *lpr*).

Powyższe ograniczone uprawnienia pozwalają znacząco zmniejszyć rozmiar szkód spowodowanych ewentualnym włamaniem wykorzystującym np. przepełnienie bufora w przeglądarce internetowej.

## Role i domeny w służbie demonów

Użycie separacji uprawnień w stosunku do programów uruchamianych przez użytkownika to jedno pole działania SELinuxa.

Jednak ze względu na fakt, że Linux wciąż jest najczęściej stosowany jako system dla serwerów, a także ponieważ to zwykle systemy serwerowe miewają najwyższe wymagania co do jakości polityki bezpieczeństwa, wydaje się, że użycie SELinuxa dla rozdzielenia uprawnień pomiędzy różnorodne serwisy, demony itd. jest drugim i znacznie szerszym polem zastosowań.

Bazując na poprzednim przykładzie można już domniemywać (i tak jest w istocie), że role i domeny

dla procesów serwerowych: separują poszczególne demony, a w miarę potrzeb także same funkcje demonów i robią to lepiej, niż z użyciem SUIDów i użytkowników systemowych, gdyż:

- *policy* SELinuxa daje większą elastyczność w doborze uprawnień,
- nie trzeba tworzyć większej ilości użytkowników systemowych, by rozdzielić funkcje,
- jeśli proces serwera potrzebuje cały czas do czegoś prawa roota, to tworzenie użytkowników nie ma sensu, niejednokrotnie autorzy oprogramowania uciekają się wtedy do jeszcze bardziej skomplikowanych rozwiązań, jak chrooty i inne.

Niestety, ze względu na specyfikę rozwiązań wewnątrz różnych pakietów oprogramowania serwerowego elementy *policy* SELinuxa muszą być definiowane specjalnie dla każdego oprogramowania (lub rozszerzane w ramach klasy oprogramowania o wpisy potrzebne do poprawnego działania jej konkretnej implementacji).

Serwer *Bind* jest przykładową implementacją dla klasy oprogramowania *Serwery DNS*. Potrzebuje on praw roota co najmniej do startu, a częstokroć także do działania (jeśli musi bindować się do podnoszonych podczas jego działania dynamicznych Interfejsów). W tym wypadku niezależnie od tego, czy działa on z prawami roota, czy też zwykłego użytkownika - będziemy chcieli ograniczyć jego uprawnienia do następujących:

- bindowanie do portu 53 TCP i UDP,
- odczyt plików konfiguracyjnych i plików domen,
- zapis tylko do plików cache.

Dzięki temu nawet w wypadku włamania do tego demona gdy działa on jako root atakujący praktycznie nie będzie miał możli-

## Słowniczek

- *RBAC* – system kontroli dostępu oparty na rolach,
- *DTAC* – mechanizm (niskopoziomowy) definiowania zasad dostępu używany przez SELinuxa do realizacji *RBAC*,
- *typ* – atrybut każdego obiektu dostępu używany jako jedyny do określenia praw dostępu do niego z danej domeny,
- *użytkownik* – odpowiada użytkownikowi w sensie uniksowym, jednak w SELinuxie jeden użytkownik może mieć przypisanych wiele ról,
- *rola* – jedna z funkcji wykonywanych przez użytkowników, do jej skutecznego wykonania może być potrzebna praw do jednej lub wielu domen,
- *domena* - definiuje zestaw praw do obiektów o wskazanych typach
- *policy* – zestaw reguł w systemie SELinux,
- *security context* - kontekst bezpieczeństwa, dotyczy zarówno obiektów (dla nich znaczenie ma tylko *typ*), jak i procesów.

wości zniszczenia żadnych danych w systemie (będzie można co najwyżej odsyłać nieprawdziwe odpowiedzi na zapytania Innych serwerów DNS).

## Definicje Type Enforcement w SELinux policy

Definicje TE w *policy* dotyczą: deklaracji typów, wymuszonej zmiany typu obiektu (np. pliku na dysku), dozwolonej zmiany typu (np. pliku urządzenia konsoli) czy tzw. *wektorów dostępu* do wszelkiego typu obiektów. Przykłady tych definicji znajdują się na Listingu 1.

Typy, które będą używane przez inne reguły w *policy*, muszą zostać najpierw zdefiniowane. W pierwszej części listingu widać definicje typów: domeny *sshd\_t* (główna domena używana przez programy z pakietu *SSH*) i typów plików używanych przez programy działające w tej domenie.

Dzięki tym dodatkowym typom będzie rozdzielić np. uprawnienia do plików tymczasowych *SSH* (typ: *sshd\_tmp\_t*) od innych plików tymczasowych (zwykle mających typ *tmp\_t*).

W drugiej części listingu widać definicje wymuszonych przejść. W tym wypadku definicja jest bardziej czytelna dzięki zastosowaniu makr, np. *file\_type\_auto\_trans*. Mechanizm makr w SELinux *policy* znakomity sposób ułatwia proces tworzenia lub modyfikacji *policy*. W tym wypadku *file\_type\_auto\_trans* mówi, że gdy proces w domenie *sshd\_t* otworzy (np. jednocześnie go tworząc) plik, który normalnie miałby domenę *tmp\_t*, to nastąpi automatyczna zmiana domeny pliku na *sshd\_tmp\_t*.

Z kolei pierwsze makro *domain\_auto\_trans* mówi, że jeżeli proces działający w domenie *initrc\_t* (a w takiej domenie są uruchamiane wszystkie skrypty w */etc/init.d*) uruchomi program z pliku o typie *sshd\_exec\_t*, to nowo uruchomiony program będzie działał w domenie *sshd\_t*. Taka deklaracja (I specjalny typ *\*\_exec\_t*) jest konieczna praktycznie dla każdego pakietu oprogramowania uruchamianego w */etc/init.d/*

W trzeciej części listingu umieszczone są zezwolenia na zmianę typu obiektu dokonywanego przez działający w pewnej domenie program. Na przykład program działający w domenie *user\_t* będzie miał prawo zmienić typ pliku urządzenia znakowego z *tty\_device\_t* na *user\_tty\_device\_t*.

Widać tutaj, że w szczególny sposób potraktowane są konsole, na których ma działać użytkownik w domenie *sysadm\_t* – będzie on mógł zmienić typ pliku urządzenia znakowego (konsoli, z której ma zamiar korzystać) na *sysadm\_tty\_device\_t*. Ma to dodatkowo utrudnić ewentualne podsłuchiwanie lub modyfikowanie komunikacji dokonywanej z użyciem tego urządzenia gdy jest ono używane przez proces w domenie *sysadm\_t*.

W ostatniej części listingu zawarte są szczegółowe informacje o tym, co proces działający w danej domenie może zrobić z obiektem danego typu.

**Listing 1.** Definicje Type Enforcement w SELinux policy  
Deklaracje typów:

```
type sshd_t, domain, privuser, privrole, privlog,
    privowner;
type sshd_exec_t, file_type, exec_type, sysadmfile;
type sshd_tmp_t, file_type, sysadmfile, tmpfile;
type sshd_var_run_t, file_type, sysadmfile, pidfile;
```

Definicje *wymuszonych przejść* (makra):

```
domain_auto_trans(initrc_t, sshd_exec_t, sshd_t)
file_type_auto_trans(sshd_t, tmp_t, sshd_tmp_t)
domain_auto_trans(sshd_t, shell_exec_t, user_t)
```

Definicje *dozwolonych przejść* (bez makr i z makrem):

```
type_change user_t tty_device_t:
    chr_file user_tty_device_t;
type_change sysadm_t tty_device_t:
    chr_file sysadm_tty_device_t;
type_change user_t sshd_devpts_t:
    chr_file user_devpts_t;
type_change sysadm_t sshd_devpts_t:
    chr_file sysadm_devpts_t;
domain_trans(sshd_t, shell_exec_t, sysadm_t)
```

Definicje *wektorów dostępu*:

```
allow sshd_t sshd_exec_t:file {
    read execute entrypoint };
allow sshd_t sshd_tmp_t:file {
    create read write getattr setattr link unlink
        rename };
allow sshd_t user_t:process transition;
```

## Definicje Role Based Access Control w SELinux Policy

Drugim rodzajem elementów w *policy* SELinuxa są wpisy odnoszące się do mechanizmu RBAC. Ich przykłady zostały umieszczone na Listingu 2.

Ponieważ, jak nazwa głosi, jest to system oparty na rolach, to nie może zabraknąć definicji ról. W tym wypadku definiujemy również z jakich domen kiedykolwiek dana rola może korzystać. Na przykład dla roli *user\_r* dostępna jest podstawowa, domyślna domena *user\_t*, a także pomocnicze domeny *user\_netscape\_t* i *user\_irc\_t*, które zostały omówione wcześniej.

W pewnych okolicznościach przydatna jest możliwość przełączania się między rolami na życzenie (służy do tego program *newrole*). Przykłady dozwolonych przejść między rolami zawarte są w drugiej części Listingu 2.

Generalnie rzecz biorąc system RBAC jest mocno niezależny od kont użytkowników uniksowych. Jednak w wielu przypadkach pożądane jest powiązanie konkretnych użytkowników z zestawami ról, które są dla nich dozwolone. Na załączonym przykładzie widać np., że użytkownik *root* domyślnie występuje w roli *user\_r*, chociaż ma również dostęp do roli *sysadm\_r*.

## Security Context obiektów sieciowych

Ciekawą możliwością jest definiowanie *Security Context* dla obiektów systemu związanych z siecią. W jasny sposób pokazuje to, że w rzeczywistości obiekty w systemie posiadają pełny SC, pomimo że używany jest tylko typ. W pierwszej części Listingu 3 widać, że zdefiniowano typ *http\_port\_t*, który będą posiadały porty TCP o numerach 80 i 8080. Dzięki temu w innych elementach *policy* będzie można zdefiniować, że dostęp do tych portów, a dokładniej do ich typu (np. możliwość bindowania się do nich), mają mieć tylko procesy z wybranych domen. Będzie to np. domena *httpd\_t*, w której będzie działał serwer WWW uruchomiony z binarium o typie *httpd\_exec\_t*, dla którego będzie zdefiniowane wymuszone przejście podczas uruchomienia - do domeny *httpd\_t* właśnie. W podobny sposób można regulować prawa do interfejsów sieciowych.

## Domyślne Security Contexts plików

Wiadomo, że każdy plik w systemie plików będąc obiektem ma nadany SC, a w nim typ. Przyjrzyjmy się tej kwestii nieco bliżej. SC plików (a przede wszystkim ich typy) nie są nadawane w sposób automatyczny w momencie uruchomienia systemu z jądrem SELinuxa. Muszą one być nadane w procesie *etykietowania* (ang. *labelling*), który musi zostać przeprowadzony wcześniej, zanim zacniemy używać systemu z SELinuxem. Pytanie jakie należy sobie zadać brzmi: *skąd wiadomo, jaki typ powinien mieć dany plik?*

Gdybyśmy zainstalowali źródła SELinuxa to odpowiedź można by znaleźć w pliku o nazwie *file\_contexts*,

**Listing 2.** Definicje Role Based Access Control w SELinux Policy

Definicje ról:

```
role system_r types {
    kernel_t initrc_t getty_t klogd_t };
role user_r types {
    user_t user_netscape_t user_irc_t };
role sysadm_r types { sysadm_t run_init_t };
```

Definicje dozwolonych przejść:

```
allow system_r { user_r sysadm_r };
allow user_r sysadm_r;
allow sysadm_r system_r;
```

Definicje rozpoznawanych użytkowników:

```
user system_u roles system_r;
user root roles { user_r sysadm_r };
user jdoe roles user_r;
```

### Listing 3. Definicje kontekstów bezpieczeństwa (SC) obiektów systemu

Definicje kontekstów sieciowych - porty:

```
portcon tcp 80 system_u:object_r:http_port_t
portcon tcp 8080 system_u:object_r:http_port_t
```

Definicje kontekstów sieciowych – interfejsy sieciowe:

```
netifcon eth0 system_u:object_r:netif_eth0_t
                system_u:object_r:netmsg_eth0_t
netifcon eth1 system_u:object_r:netif_eth1_t
                system_u:object_r:netmsg_eth1_t
```

Definicje domyślnych kontekstów plików:

```
/home system_u:object_r:home_root_t
/home/[^/]+ -d system_u:object_r:
    user_home_dir_t
/home/[^/]+/.+ system_u:object_r:
    user_home_t
/home/[^/]+/.ssh(/.)*? system_u:object_r:
    user_home_ssh_t
```

stanowiącym składową *policy*. Zawiera on definicje, których przykład znajduje się w drugiej części Listingu 3. Informują one program przeprowadzający etykietowanie jakie powinien nadać typy (a dokładniej *konteksty bezpieczeństwa*) różnym plikom w systemie. W podanym przykładzie widać, że katalog `/home` otrzyma typ `home_root_t`, katalogi użytkowników – `user_home_dir_t`, a znajdujące się w nich pliki i katalogi – typ `user_home_t`. W sposób szczególny potraktowane zostaną pliki używane przez SSH, a znajdujące się w katalogach domowych użytkowników. Otrzymają one własny typ `user_home_ssh_t`. Podobne definicje istnieją dla katalogów używanych przez np. klientów IRC, gdzie trzymane są ich logi i konfiguracja.

## Instalacja SELinuxa

Mając już podstawową wiedzę na temat SELinuxa najwyższy czas przystąpić do jej praktycznego wdrożenia. Aby móc używać SELinuxa trzeba zgromadzić potrzebne elementy:

- łąty na jądro z LSM i SELinuxem,
- łąty na pewne istotne programy w systemie (*login*, *ssh*, *ls*, *ps*, *xdm*,...),
- zestaw programów narzędziowych specyficznych dla SELinuxa (*chcon* i inne).

Zamiast kłopotać się z łątami można ze strony NSA pobrać pełne źródła zmodyfikowanego jądra i zmodyfikowanych programów systemowych. Jeżeli korzystamy z dystrybucji Debiana (stabilnej bądź niestabilnej), to są dostępne źródła APT zawierające wszystko, co jest potrzebne do instalacji i użytkowania SELinuxa.

W załatanym jądrze należy włączyć opcje wskazane w Ramce. Ponieważ i tak musimy kompilować jądro samodzielnie, to najlepiej nie korzystać z obrazu *initrd* – inaczej po każdej zmianie *policy* będzie trzeba nową *policy* umieścić również w obrazie *initrd*, a to jest zawsze kłopotliwe i czasochłonne.

Po instalacji jądra i programów użytkowych zmodyfikowanych do potrzeb SELinuxa trzeba jeszcze przeprowadzić operację *etykietowania*. Jednak przed jej przeprowadzeniem należy sprawdzić, czy ścieżki podane w pliku *file\_contexts* są odpowiednie dla naszej dystrybucji. Jeżeli korzystamy ze źródeł pobranych ze strony NSA, to domyślnie wpisy są odpowiednie dla dystrybucji RedHat. Jeżeli korzystamy z Debiana i pobraliśmy pakiety z odpowiedniego źródła APT, to są one już odpowiednio poprawione. Użytkownicy pozostałych dystrybucji powinni koniecznie sprawdzić zawartość tego pliku. Potem można już wykonać operację etykietowania. W dystrybucji SELinuxa znajduje się plik *Makefile*, który zawiera następujące cele:

- *install*- kompilacja i instalacja *policy*,
- *load*- kompilacja, Instalacja i załadowanie *policy*,
- *reload* - kompilacja, instalacja i załadowanie (przeładowanie) *policy*,
- *relabel*- reetykietowanie systemu plików (FCs),
- *policy*- lokalna kompilacja *policy* do testów.

Operacja etykietowania może trwać dość długo, ponieważ wymaga wykonania operacji na każdym z plików w lokalnych systemach plików.

Po jej zakończeniu można zauważyć, że w katalogu głównym każdej partycji powstał katalog o nazwie *...security* zawierający kilka plików binarnych - właśnie z dodatkowymi informacjami o każdym z plików na partycji. Mając zainstalowane jądro z SELinuxem i programy do niego dostosowane można przystąpić do testów praktycznych.

## Pierwszy start

Nawet jeśli wszystko zostało wykonane z należytą starannością, to z pewnością podczas pierwszego startu pojawi się wiele komunikatów o błędach zgłaszanych przez rozszerzenie SELinuxa. Mimo to system powinien w całości wystartować (dlaczego - o tym za chwilę). Przykładowy (jeden!) komunikat o błędzie jest pokazany na Listingu 4.

Informuje on, że proces `/bin/login` posiadający SC `system_u:system_r:local_login_t` próbował zapisywać do

### Listing 4. Przykładowy komunikat o błędzie podczas uruchamiania SELinuxa

```
avc: denied { write } for pid=1112
    exe=/bin/login path=/dev/log dev=03:01 ino=15
    scontext=system_u:system_r:local_login_t
    tcontext=system_u:object_r:device_t tclass=sock_file
```

### Opcje, które należy włączyć w załatanym jądrze

#### Networking Options

- [Y] Network Packet Filtering

#### Security Options

- [Y] Enable different security models
- [Y] Capabilities Support
- [Y] NSA SELinux Support
- [Y] NSA SELinux Development Support

gniazda `/dev/log` o typie `device_t`. Komunikat ten wystąpił w rzeczywistej sytuacji i wynikał z niedostosowania `policy` do używanego przeze mnie demona `syslog-ng`. Po dodaniu do `policy` dwu linii (podanych mi przez Russella Cokera, dewelopera Debiana i SELinuxa) - problem ustąpił.

Dzięki zaznaczeniu podczas konfiguracji jądra opcji `NSA SELinux Development Support` - SELinux startuje w trybie `permissive`, w którym to wszystkie błędy wynikające z zabronionego dostępu są zgłaszane, ale sam dostęp do obiektów nie jest zabraniany. System działa wtedy tak, jak normalny Linux, ale zgłasza błędy. Istnieje kilka metod na przełączenie do trybu `enforcing`, kiedy to SELinux rzeczywiście realizuje zasady zawarte w `policy`:

- wyłączenie `NSA SELinux Development Support`,
- przełączenie poleceniem `avc_toggle` (jeśli zezwala na to `policy` - można np. nie pozwolić na powrót do trybu `permissive`),
- podanie jądra (przy starcie) parametru `permissive=1`.

System można startować w trybie `enforcing` dopiero po upewnieniu się, że zasady `policy` są odpowiednio dostosowane i że system będzie w stanie działać tym trybie. Nie należy próbować eliminować wszystkich zgłaszanych błędów, gdyż nie wszystkie one są krytyczne i czasami wynikają w pełni z zamierzonej polityki bezpieczeństwa.

Należy w szczególności zwrócić uwagę na procesy działające (lub próbujące działać) w domenie `initrc_t` (`ps -e --context` pokazuje konteksty procesów). Oznacza to, że najprawdopodobniej dla pakietu oprogramowania, do którego należy dany program, brakuje odpowiednich wpisów w `policy` i podczas startu ze skryptów `/etc/init.d/` `SC` nie jest zmieniany na właściwy dla danego pakietu oprogramowania. Należy też skontrolować `file_contexts`, czy przypadkiem ścieżki nie są nieprawidłowe.

Jeżeli dany pakiet oprogramowania w ogóle nie jest objęty wpisami w `policy`, to najlepiej zwrócić się o pomoc w ich napisaniu do deweloperów SELinuxa. Pewną pomocą może też być czasami skrypt `scripts/newrules.pl`, który stara się generować nowe zasady `allow na` podstawie zgłaszanych błędów. Po starcie systemu będzie można się zalogować. Jeżeli

spróbujemy zalogować się jako `root` z lokalnej konsoli, to będziemy mieli bezpośredni dostęp `c:` roli `sysadm_r` (która w odróżnieniu od domyślnej `user_r` daje rzeczywiste możliwości administrowania systemem). Fragment ekranu logowania przedstawiono na Listingu 5.

Rolę, z którą działa `root`, można też zmienić później (w ramach dozwolonych przez `policy`) za pomocą polecenia `newrole -r sysadm_r`. Dostęp do roli `sysadm_r` zwykle wymaga ponownego podania hasła `roota`.

## Przydatne informacje

Jeżeli korzystamy z któregoś z graficznych menedżerów. logowania, jak `xdm`, `gdm` czy `kdm`, to musi on być załadowany aby działał pod SELinuxem. Jeżeli nie chcemy go łączyć to zawsze można użyć `startx`, startując konsolę graficzną z konsoli tekstowej (login już wykonał zmianę domen, którą inaczej musiałby wykonać `*dm`).

Dobrze jest wiedzieć, czy działamy w trybie `enforcing`, czy `permissive`. Można to sprawdzić poleceniem `avc_enforcing`. Demony, które w trybie `permissive` startują w domenie `initrc_t`, prawie na pewno nie będą poprawnie działać w trybie `enforcing`. Trzeba poprawić `policy`, aby działały we własnych domenach.

Przydatne polecenia to `ps -e --context` i `ps --context`. Pierwsze z nich pozwala sprawdzić, z jakimi `SC` działają procesy. Drugie z nich - jakie typy mają pliki. Do celów testowych można zmieniać `SC` plików za pomocą polecenia `chcon`. Należy pamiętać, że jeśli nie zostaną dodane odpowiednie wpisy do `file_contexts`, to zmiana ta zostanie zniwelowana po kolejnym etykietowaniu.

Jeżeli przełączamy się między jądrem skompilowanym z SELinuxem a jądrem bez niego, to przed ponownym włączeniem jądra z SELinuxem należy ponownie przeprowadzić etykietowanie, ponieważ w międzyczasie mogą powstać pliki nie posiadające `SC`. Chociaż nic nie stoi na przeszkodzie, aby cały czas używać jądra z wkompilem SELinuxem, ale w trybie `permissive`.

`Policy` jest komponowane z wielu elementów, z których każdy zazwyczaj odpowiada za obsługę jednego pakietu oprogramowania, np. serwera DNS. Jednak w większość przypadków w naszym systemie nie ma zainstalowanych wszystkich pakietów oprogramowania, dla których są dostępne elementy `policy`.

Dlatego np. podczas instalacji pakietów dla Debiana zostaniemy zapytani, których z elementów `policy` chcemy rzeczywiście używać. Używając wszystkich dostępnych elementów `policy` będziemy mieli około 200 tysięcy zasad.

### Przydatne polecenia

- `avc_toggle` - przełącza `permissive/enforcing`
- `avc_enforcing` - sprawdza bieżący tryb
- `ps -e --context` - wyświetla `SC` procesów
- `ls --context` - wyświetla `SC` plików
- `chcon` - pozwala zmienić `SC` plików

**Listing 5.** Ekran logowania użytkownika root

```
Your default context is root:user_r:user_t
Do you want to choose a different one? [n]y
[1] root:user_r:user_t
[2] root:sysadm_r:sysadm_t
Enter number of choice: 2
```

Być może warto tę liczbę zmniejszyć do kilkudziesięciu tysięcy zasad rzeczywiście potrzebnych.

## Berlin, patenty i ogólnodostępny root

Muszę w tym miejscu zmartwić fanów konsoli graficznej. Niestety, SELinux nie jest w stanie w znaczący sposób poprawić problemów z bezpieczeństwem wynikających z obecnego standardu X protokołu, a co za tym idzie – X window. Problem polega na tym, że nawet jeśli nasz xterm, klient IRC czy przeglądarka Internetowa działają w osobnych domenach, to nadal wszystkie te programy mają możliwość podglądania zawartości okien Innych programów i podglądania klawiatury.

Dlatego też błąd w kliencie IRC nadal wystawia nasz system na niebezpieczeństwo. Gdybyśmy np. chcieli z terminala zalogować się gdzieś, to atakujący będzie mógł poznać (podstuchać) nasze hasło. Być może w przyszłych wersjach X protokołu zostaną wprowadzone np. dwa poziomy zaufania do programów: niższy, bez wyżej opisanych uprawnień, i wyższy, taki jak dotychczasowy. Jednak nacisk na takie zmiany wydaje się być niewielki. Projekt *Berlin* alternatywnego protokołu i serwera, który m.in. zawierał elementy bardziej zaawansowanej polityki bezpieczeństwa - niestety nie porwał za sobą rzesz fanów czy deweloperów i nie zyskał popularności, a co za tym idzie praktycznie nie jest używany.

Jest jeszcze druga sprawa - patentów. W tej chwili NSA finansuje rozwój SELinuxa, ale patenty na niego posiada komercyjna firma. Może się rodzić podejrzenie, czy firma ta nie zechce w pewnej chwili pieniędzy za użytkowanie oprogramowania korzystającego z opatentowanych przez nią rozwiązań. NSA wcale nie ma zamiaru np. wykupywać tych patentów (co stworzyłoby niemiłą precedens), ani też zapewne *Secure Computing Corp.* nie bardzo chce walczyć z potężną agencją rządową (bo to ona byłaby pierwszym celem). Jednak firma ta jak dotąd nie złożyła jasnej deklaracji udostępnienia patentu na licencji GPL, co powoduje, że SELinux najprawdopodobniej nie wejdzie do jądra 2.5 i 2.6. Jeśli kwestie licencyjno-patentowe zostaną wyjaśnione, to oprogramowanie to powinno znaleźć się w pierwszych wersjach 2.7.

Ciekawy pokaz możliwości SELinuxa przeprowadził Russell Coker – deweloper Debiana, a także jeden z nielicznych nie opłacanych przez nikogo deweloperów SELinuxa. Otóż udo-

stąpił on w Internecie maszynę z publicznie znanym hasłem roota.

Okazało się, że będąc rootem można było np. wykonywać chroot, ale praktycznie niczego w systemie nie dało się posuwać. Zastosowane wpisy w policy nie były wcale bardziej restrykcyjne od standardowych, a wręcz zostały w pewnych miejscach złagodzone, np. aby każdy mógł odczytać logi systemowe i wiedzieć, dlaczego nie mógł wykonać danego działania. Myślę, że dobrze świadczy to o sile pomysłu SELinuxa.

## I co dalej? Używać!

Russell Coker, na pytanie o zakres zastosowania SELinuxa odpowiedział, że stosuje go wszędzie: zarówno na swoich stacjach roboczych, laptopie, jak i na serwerach. Ja jednak myślę, że SELinux przede wszystkim powinien być i będzie szczególnie często używany w systemach serwerowych, gdzie wymagania dotyczące bezpieczeństwa są niejednokrotnie bardzo wysokie. W chwili obecnej SELinux wychodzi z ukrycia. Rysują się następujące możliwości rozwoju i upowszechnienia tej technologii:

- obecność w oficjalnym jądrze - LSM już są obecne w gałęzi 2.5, a SELinux jest (objętościowo, w porównaniu do LSM) jedynie niewielkim dodatkiem, który NSA chce koniecznie zintegrować; jeśli tylko kwestie licencyjno-patentowe zostaną wyjaśnione, to może nawet doczekamy się SELinuxa w późniejszych wersjach 2.6?
- rozbudowanie listy pakietów oprogramowania obsługiwanych przez standardową *policy* (tak, aby administrator nie musiał pisać reguł)
- integracja z dystrybucjami - najlepiej tutaj radzi sobie Debian, w którym dzięki wysiłkom Russella Cokera część programów już potrafi radzić sobie z SELinuxem, a w przyszłości SELinux ma stać się standardową (choć opcjonalną) częścią dystrybucji Debiana

Nie należy ulegać złudzeniu, że technologia ta dopiero będzie zasługiwała na zainteresowanie. Już w chwili obecnej SELinux jest w pełni sprawnym, działającym, przetestowanym i elastycznym narzędziem podwyższania bezpieczeństwa systemów linuksowych. Posiada sprawne zaplecze aktywnych deweloperów, wkrótce będzie zintegrowany ze standardowym jądrem i popularnymi dystrybucjami. Z pewnością błędem byłoby zwlekać z poznaniem tak fascynującej i użytecznej technologii! ■

### W Sieci

- NSA (download, dokumentacja, listy dyskusyjne) - <http://www.nsa.gov/selinux>
- Debian Stable - wpis w *sources.list*:  
*deb http://www.microcomaustalia.com.au/debian/stable selinux*
- Debian Unstable - wpis w *sources.list*:  
*deb http://www.coker.com.au/selinux/*