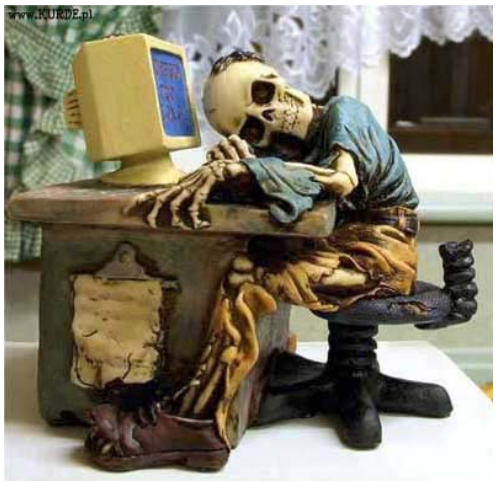


Co łączy potężną Narodową Agencję Bezpieczeństwa, posiadającą budżet większy niż polskie wojsko z programistami z całego świata działającymi w projektach typu open-source?

Odpowiedź: SELinux

Jeden z guru SELinuxa (Security Enhanced Linux – Linux z rozszerzonym bezpieczeństwem), osoba która dostosowała go do Debiana, opisał okoliczności jego powstania jako sytuację, w której Papież schodzi z balkonu w Rzymie, częstuje ludzi chlebem i rybami, a potem zaprasza ich do siebie do domu na mecz piłki nożnej i pare piw.

Bo koniec końców niespotykana jest sytuacja kiedy równie tajna, chociaż mniej sławna niż CIA agencja rządowa, finansuje projekt badawczy kosztem milionów dolarów tylko po to, aby po fakcie udostępnić go wszystkim internautom, a nawet doprowadzić do tego aby stał się częścią jądra 2.6. Co prawda wszystkie patenty zatrzymała firma, która ów projekt prowadziła, lecz nikt nie wierzy żeby nagle mogła zarządać zapłaty – w końcu pierwszym jej adwersarzem musiałaby być NSA.

Czym jest SELinux?

SELinux jest dodatkiem do jądra, swoistą nakładką na istniejącą filizofię bezpieczeństwa (profesjonalnie można powiedzieć, że jest implementacją Linux Security Module, który za to jest cześcią architektury jądra oferującą wsparcie dla różnych modeli bezpieczeństwa).
Nie jest on odrębną dystrybucją, prawdę mówiąc można go już znaleźć jako opcję w każdej szanującej się dystrybucji.

Projekt ten został podjęty aby zademonstrować środowisku użytkowników i pasjonatów Linuxa znaczenie i wartość ścisłej kontroli dostępu procesów do różnych zasobów oraz jak takową kontrolę do Linuxa dodać. Kernel SE Linux zawiera nowe architekturowo komponenty, napisane i rozwijane w ramach tego projektu aby zwiększyć bezpieczeństwo systemu operacyjnego typu Flask (**Fl**ux **A**dvanced **S**ecurity **K**ernel – system o takiej architekturze pojęcia bezpieczeństwa, która oferuje elastyczne wsparcie dla różnych polityk bezpieczeństwa). Te komponenty dają ogólne wsparcie do wprowadzania różnych rodzajów polityk tzw. obowiązkowej kontroli dostępu (MAC).

Projekt jest rozwijany przez firmę Secure Computing Corp. od 1992 roku (oczywiście wtedy trochę inaczej się nazywał i nie był związany z Linuxem). SELinux jest potomkiem projektu DTMach (Distributed Trusted Machine) którego dzieckiem był system operacyjny Fluke. Równolegle prowadzony projekt Flux rozwijał architekturę dla tego systemu – w/w wymienioną Flask. Gdy osiągnieto duże efekty, uznano że czas zintegrować Flask z systemem troszkę bardziej znanym niż „sławny" Fluke. Wybrano Linuxa, dlaczego?

Jak samo NSA tłumaczy, Linux został wybrany dlatego, że stale osiąga coraz większe sukcesy i popularność i co najważniejsze, jest rozwijany systemem open-source, co umożliwia projektowi SELinux bycie sprawdzanym,poprawianym i rozwijanym przez największe możliwe grono informatyków. Ponadto Linux zapewnia idealną szansę na pokazanie że nastawienie na bezpieczeństwo może być efektywnie sprzężone z popularnym systemem operacyjnym.

Ale suma sumarum pozostaje pytanie jaki ma w tym interes NSA? Oczywiście możemy się tylko domyślać, chociaż można mieć pewność, że część rozwiązań jest stosowana do rozwijania już w pełni tajnego projektu bezpiecznego systemu operacyjnego, który mógłby być używany przez instytucje państwowe. Samo NSA na pytanie czy stosuje cześć rozwiązań, odpowiada że oczywiście nie może ujawnić, zatem stosuje.

Nie można też zapominać, że jednym ze statutowych obowiązków NSA jest dbanie o bezpieczeństwo USA, a więc także bezpieczeństwo biznesu amerykańskiego. Udostępnienie rozwiązań mogących lepiej zabezpieczać systemy może zmniejszyć straty firm spowodowane włamaniami do systemów, szacowane na około 60 mld. dolarów rocznie!

SELinux to system z MAC, który realizuje RBAC, za pomocą DTAC

Wszystko jasne?

SELinux wprowadza politykę obowiązkowej kontroli dostępu (MAC), która ogranicza programy użytkownika oraz usługi systemowe do obszaru systemu w którym mają minimalną liczbę przywilejów, których potrzebują do wykonania swoich zadań. Procesy zamknięte w ten sposób mają zredukowaną lub wyeliminowaną możliwość czynienia szkód w systemie w wyniku wymknięcia się ich spod kontroli uzytkownika i systemu (np.: poprzez wykorzystanie błędu przepełnionego bufora lub błędną konfigurację. Ten mechanizm ograniczania działa niezależnie od tradycyjnych mechanizmów kontroli dostępu systemu Linux. Nie ma w nim żadnej koncepcji uzytkownika o wiekszych przywilejach (root, superuser), ani żadnych niedociągnięc mechanizmów bezpieczeństwa tradycyjnego Linuxa (takich jak zależność od setuid/setgid).
Bezpieczeństwo niezmodyfikowanego Linuxa zależy od poprawności jądra, uprzywilejowanych aplikacji oraz ich konfiguracji. Błąd lub luka w którymkolwiek z tych obszarów systemu może umożliwić komuś z zewnątrz przejęcie kontroli nawet nad całym systemem. Dla porównania, bezpieczeństwo zmodyfikowanego jądra opartego na SE Linux zależy głównie od poprawności jądra oraz konfiguracji jego polityki bezpieczeństwa. W takim przypadku błędy lub luki w aplikacjach czy ich konfiguracjach mogą pozwolić na ograniczone niewłaściwe działanie pojedynczych programów czy demonów systemowych, jednakże nie stanowią groźby dla bezpieczeństwa innych programów i komponentów systemu, ani dla systemu jako całości.

Ale co dokładnie znaczą te tajemnicze skróty?

MAC to jak już zostało napisane obowiązkowa kontrola dostępu (Mandatory Access Control). Jest to technika, która rozszerza kontrolę dostępu opartą na uprawnieniach systemu plików oraz koncepcji użytkowników i grup. Najważniejsza zasada w filozofii MAC brzmi:

użytkownik nie decyduje o zabezpieczeniach i prawach dostępu do obiektu. Owe zabezpieczenia i prawa są definiowane odgórnie przez administratora, zgodnie z polityką bezpieczeństwa, którą zdecyduje się on realizować. Tradycyjne systemy kontroli umożliwiają użytkownikowi w całości określić uprawnienia swoich zasobów, co oznacza że mogą oni przez przypadek albo przemyślne działanie, umożliwić nieautoryzowanemu użytkownikowi dostęp do tych zasobów.
MAC zakłada też, że przed każdą operacją proces, który chce ją wykonać musi dostać pozwolenie od instancji sprawującej kontrolę nad dostępami do różnych zasobów (w SELinux jest to realizowane w ten sposób, że funkcja kontrolująca otrzymuje tzw. kontekst bezpieczeństwa procesu oraz obiektu, na którym ów proces chce wykonać jakieś czynności; funkcja ta opierając się na regułach polityki bezpieczeństwa, decyduje czy proces uzyska dostęp czy nie).
Celem MAC jest także stworzenie systemu wielopoziomowego bezpieczeństwa (umożliwienia w pełni bezpiecznego użytkowania tego samego systemu przez użytkowników o różnych poziomach autoryzacji dostępu do róznych typów informacji z róznymi wymaganiami bezpieczeństwa, bez konieczności ujednolicania tych wymagań)

RBAC (Role Based Access Control) – system kontroli dostępu oparty na rolach. Logika działania RBAC jest podobna do logiki działania firmy. Rózni ludzie mają w niej różne role. Role się różnią między sobą bo Ci ludzie wykonują różne prace, mają różne zadania. Wypełniając daną rolę, mają prawo do używania pewnych zasobów, których nie mogą (bo tego nie potrzebują) używać inni i vice versa.

Zatem użytkownik w systemie ma przypisaną jakąś rolę z puli ról jemu dostępnych. Naraz może wykonywać tylko jedną rolę, lecz w każdej chwili może się między nimi przełączać. Usprawnia to mechanizm nadawania uprawnień – nie są one przypisywane do użytkownika tylko do roli. Można zatem tworzyć takie pakiety praw dostępu do różnych obiektów i w zależności od poziomu autoryzacji użytkowników przyznawać im te pakiety w postaci ról.

Jakby dłużej się zastanowić, RBAC w sumie nie jest niczym nowym – istnieje w standardowym Unix-ie od zarania dziejów: mamy pewnych użytkowników systemowych, z których każdy ma prawo wykonywać pewien zakres działań (realizować pewną rolę w systemie). Ponadto poprzez istnienie SUID-ów możemy przełączać się między tymi rolami.

A więc niby już mamy to co zakłada RBAC, lecz naprawdę RBAC jest rozwinięciem tradycyjnych zabezpieczeń, bo tak naprawdę skupia się nie na nadawania praw dostępu obiektom, lecz na określaniu tychże wobec operacji między abstrakcyjnymi grupami obiektów (oczywiście grupa może być jednoelementowa i wtedy wszytsko zostaje po staremu).

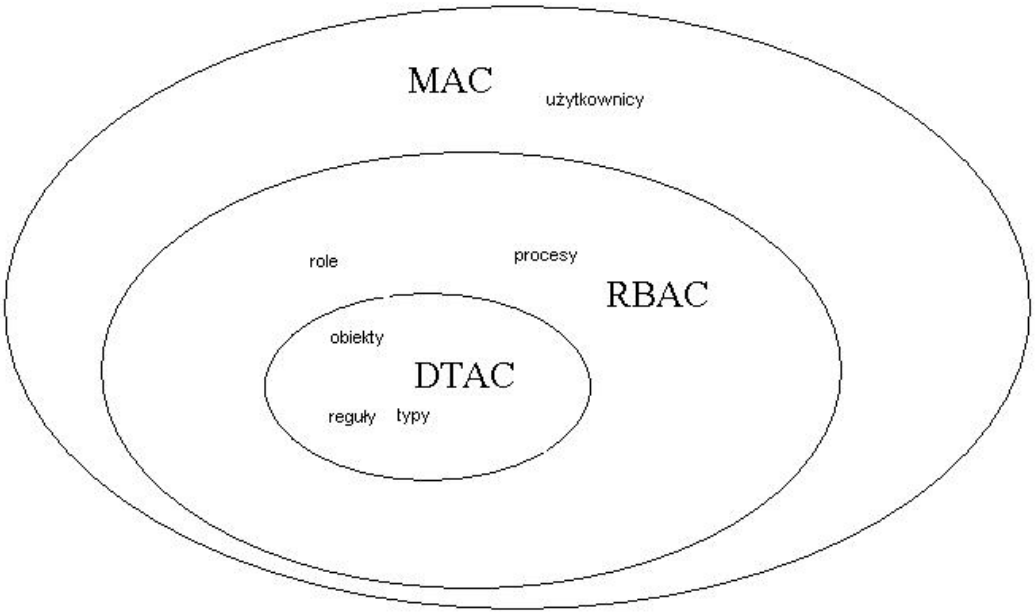
Skoro określa relacje między abstrakcyjnymi grupami, to coś musi te abstrakcje określić i służy do tego technologia **DTAC** (Dynamically Typed Access Control) – polskie tłumaczenie tego określenia byłoby potworkiem językowym więc posłużymy się adekwatnym substytutem: domenowy system kontroli dostępu.

DTAC zakłada, że każdy obiekt w systemie posiada pojedynczy typ – taki sam na całe swoje istnienie w cyberprzestrzeni. Skoro każdy obiekt to wszystkie pliki, katalogi, urządzenia, interfejsy sieciowe etc. Ten typ jest zdeterminowany przez odgórnie (czytaj: administrator) stworzone reguły przydzielania typów poszczególnym obiektom. Użytkownik zatem nie może w żaden sposób zmienić typu obiektu.

Ilość typów jest nieograniczona, ale oczywiście trzeba myśleć zdroworozsądkowo nie każdy plik musi mieć inny typ.

Dla każdego typu tworzone są zestawy reguł, które decydują o zachowaniu się systemu przy próbie wykonania jakichs czynności na obiektach danego typu.

SELinux standardowo ma wpisane ponad 100 000 takich reguł.



Co się składa na SELinux?

Można wyróżnić 3 części składowe SELinux, które dodaje on do oryginalnego Linuxa:
jądro – jądro SELinux to łata na jądro zwykłego Linuxa. Jak już wcześniej wspomniałem SELinux wykorzystuje infrastrukturę jądra LSM (Linux security modules), dedykowaną do takich modeli bezpieczeństwa. SELinux poprzez ten interfejs wymusza w systemie własną politykę bezpieczeństwa opartą na w/w technikach.

zmodyfikowane kluczowe programy – zasadniczo SELinux został napisany tak aby programy nie musiały być świadome, że działa w danym systemie (nowo powstające programy w większości są typu security-aware – świadome istnienia podwyższonych zasad bezpieczeństwa, np. producenci dostarczają własne przetestowane reguły, które można dodać do systemu aby ten program bezpiecznie działał). Pewne programy są jednak kluczowe dla bezpieczeństwa i muszą być rozszerzone o obsługę SELinux, są to np.: ssh, ls, ps, login.

zasady (policy) – zawierają zbiór standardowych reguł określających prawa dostępu i zachowanie się systemu w różnych sytuacjach. Krytyczna cześć całego systemu SELinux, ponieważ decydują o skutecznym jego działaniu. Standardowo dostarczany zbiór reguł jest często wystarczający do osiągnięcia wysokiego poziomu bezpieczeństwa, lecz czasem administrator może chcieć dopasować system do konkretnych wymagań i robi to poprzez edycję właśnie tych reguł.

Warte odnotowania jest, że SELinux jeśli zostanie źle skonfigurowany (czyli będzie miał źle napiane reguły) nie powoduje odsłonięcia systemu na łatwe ataki, bowiem pozostają nadal w mocy tradycyjne zabezpieczenia Linuxa. SELinux nie dodaje nowych uprawnień, tylko jeszcze bardziej ogranicza istniejące.

Dlaczego SELinux jest dobry? Bo jest tani i dobry.

SELinux jest:
- spójny i kompletny

Oferuje pełną wizję polityki bezpieczeństwa, nie skupia się na żadnych wycinkach systemu, tylko proponuje rozwiązania kompleksowe. Zawiera wszystkie konieczne narzędzia do uruchomienia bezpiecznego systemu, w tym co najważniejsze, standardowy zbiór reguł, dających od razu duży skok w jakości bezpieczeństwa.

- elastyczny
  Administrator może tworzyć swoje własne reguły i dopasowywać bezpieczeństwo do zaistaniałych potrzeb. Może także tworzyć wielopoziomowy system autoryzacji dostepów do zasobów.
- stale rozwijany
  SELinux jest projektem typu OpenSource, ponadto został cześcią oficjalnego jądra 2.6. To powoduje, że jest stale rozwijany przez wiele osób, a wszelki poprawki są zwykłymi łatkami na jądro.

Diabeł tkwi w szczegółach, czyli jak to dokładnie wygląda?

Poznaliśmy teorię dotyczącą MAC, RBAC i DTAC, teraz czas poznać jak zostało to wprowadzone w SELinux.

Użytkownik systemu może posiadać wiele ról, jedna z nich jest dla niego domyślna. Każda rola posiada zestaw domen dostępu – jedna z domen jest zawsze domyślna dla danej roli. W każdym momencie czasu użytkownik może wykonywać tylko jedną rolę w jednej domenie tej roli. Ale może przełączać się między dostępnymi domenami lub rolami.

Teraz przypomnijmy sobie o tym, że każdy obiekt w systemie ma określony typ. I to właśnie typy obiektów są skrzyżowane z domenami. Możliwe interakcje między daną domeną a typem są określane przez reguły (prawa zapisu, odczytu etc.).

Zatem jeśli chcemy wiedzieć jakie naprawdę uprawnienia ma dany proces, musimy wiedzieć w jakiej działa domenie (w jakiej domenie był użytkownik/proces który go uruchomił), i stąd możemy wnioskować co będzie mógł zrobić z obiektem danego typu.

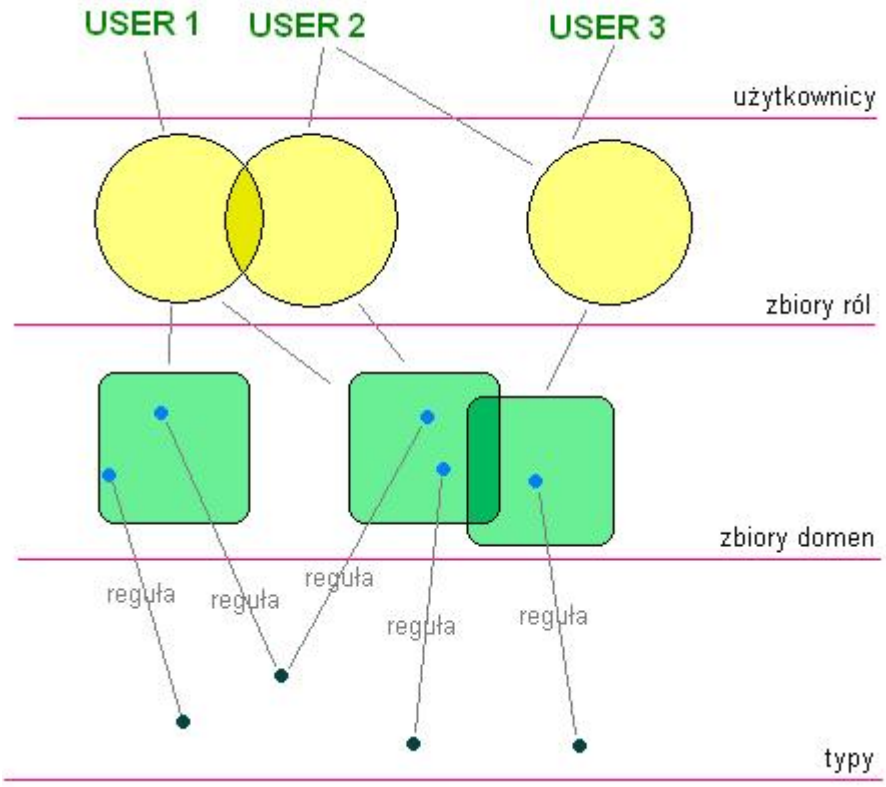
Domena może wchodzić w zestaw wielu ról, tak samo jedna rola może być wykonywana przez wielu użytkowników. Natomiast obiekt ma zawsze jeden typ.

Każdy proces i obiekt posiada swój kontekst bezpieczeństwa (security context).
Dla procesów kontekst składa się z identyfiaktora użytkownika, który go uruchomił (dla procesów systemowych jest to system_u), rola użytkownika oraz jego domena. Np.: dla użytkownika pit, który dopiero co się zalogował, kontekst powłoki będzie nastepujący:
pit:user_r:user_t
gdzie user_r i usert_t to odpowiednio domyślna rola i domena dla tej roli uzytkownika.

Obiekty także posiadają taki pełny kontekst (użytkownik, rola, typ): (system_u:object_r:typ). Jednak ważny jest jedynie typ obiektu.

Reguły najcześciej dotyczą par domena-typ. Ale oczywiście potrzebne są też inne reguły, dotyczące ról i domen. Najczęstsze reguły to:

- definiowanie operacji dozwolonych (czytanie, pisanie, tworzenie, etc.)
- dozwolone zmiany roli przez użytkownika
- dozwolone zmiany domeny w ramach roli
- automatyczna (a raczej wymuszona) zmiana domeny czy roli w przypadku zajscia jakichs konkretncyh okolicznosci (wykonanie konkretnego programu, wykonywanie operacji na konkretnym obiekcie)



Jak to w praktyce działa?

Prześledźmy działanie reguł na przykładzie klienta irc.

W związku z tym, że historia błędów w klientach irc jest długa i mroczna, w SELinux wprowadzono specjalną domenę dla użytkownika wtedy gdy chce korzystać z klienta irc.
Załóżmy, że pit jest w domenie user_t, roli user_r (a więc domyślnych). Może wykonywać ogólnodostępne programy, czytać pewne pliki. Wykonuje plik klienta irc.
Ten plik jest obiektem, a zatem ma swój typ, specjalny dla klienta irc: irc_exec_t.
Wykonanie tego pliku przez użytkownika w każdej domenie, rozpocznie się od wymuszonego przejścia użytkownika (tak naprawde procesu) do domeny user_irc_t.
Zatem domena procesu klienta irc to: user:user_r:user_irc_t
Więc jeśli nawet ktoś przejmie kontrolę nad tym klientem to będzie mógł tylko działać w domenie user_irc_t, a tak się składa że w tej domenie może tylko pisać do logów klienta i jego plików konfiguracyjnych.

W ten sposób działa właśnie ograniczanie procesów tylko do przestrzeni absolutnie koniecznej im do działania.

Dobrze, a co z procesami systemowymi?

Dotychczas mowa była o procesach użytkownika, a co się dzieje z procesami systemowymi, różnymi demonami?
Odpowiedź jest prosta: to samo. Każdy taki proces ma swój kontekst bezpieczeństwa (użytkownikiem jest system_u) i własne domeny działania. Założenie jest takie aby jak najwięcej procesów i demonów od siebie oddzielić, a więc w praktyce prawie każdy ma rodzielne domeny, tak aby błędne działanie jednego z nich ograniczało się tylko do jak najmniejszego wycinka systemu. Oczywiście powoduje to potrzebę zdefiniowania tych

domen dla procesów chakterystycznych dla różnych implementacji czy dystrybucji Linuxa, a więc dużej pracy przy regułach. Tu z pomocą przychodzi nam fakt, że w jądrze 2.6 SELinux jest już integralną częścią, więc reguły te są określone przez dystrybutora. Praca zaczyna się gdy ktoś chce SELinux uruchomić na wcześniejszych jądrach.

Przykładowo mamy jakąs implementację serwera DNS. Żeby wystartować potrzebuje on uprawnień roota, potrzebuje ich także czasem w trakcie działania. W SELinux, otrzymuje konkretną domenę w której stale może mieć uprawnienia roota, ale jest ograniczony do kluczowych i jedynych potrzebnych jemu zasobów:

- może bindować do portu 53 TCP i UDP
- może czytać pliki konfiguracyjne i pliki domen (oczywiście nie tych z reguł)
- zapisywać do plików cache

W przypadku włamania, włamywacz może naprawdę się cieszyć, bo jego możliwością popsucia czegoś w systemie będzie np.: możliwość odsyłania nieprawdziwych odpowiedzi na zapytania innych serwerów DNS. Naprawdę „niesamowita" możliwość!

I want more, show me da code!

A teraz coś co niektóre tygryski lubią najbardziej czyli przykłady plików konfigurujących reguły domena-typ, domeny i role.

Gdzie są trzymane reguły? W pliku: /etc/security/selinux/policy.conf (czasem /etc/selinux/policy.conf). Jednak jest do tylko suma wynikowa łączenia wszystkich pomniejszych plików z regułami (.te) podczas kompilowania reguł (o tym będzie pare słów później). Edytowanie pliku policy.conf pozwala nam wprowadzać zmiany od razu, żeby je zachować warto je jednak wprowadzić w owych plikach .te aby były zapamiętane na tak długo jak chcemy (a nie do następnego kompilowania).
Pliki .te są także w katalogu /etc/security/linux w podkatalogu /domains (chociaż to oczywiście może wyglądać inaczej).

Obejrzymy zawartość kilku różnych plików. Trzeba wiedzieć, że specjalnie dla reguł opracowano język m4, pełen różnych makr, słów kluczowych etc. Żeby pisać własne reguły i rozumieć istniejące potrzebna jest zatem wprawa. Oczywiście zasady pisania reguł są na tyle dużym tematem, że można mu poświęcić kilka osobnych wykładów, dlatego skupimy się na przykładach żeby mieć ogólną wiedzę jak to wygląda i jakie ma możliwosći.

Plik attrib.te

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

Plik ten opisuje definicje typów. Powyższy fragment definiuje typ sshd_t (domena używana przez programy z pakietu SSH), jak i typy plików których mogą używać programy z tej domeny. Dzięki dodatkowym typom można oddzielić pliki wykonywalne tych programów od innych plików wykonywalnych, a przez to np.: ograniczyć domenę programu z pakietu SSH w stosunku do innych programów. Atrybuty występujące po nazwach, określają:
Dla sshd_t kolejno:

- ten typ(domena) może być przypisany procesom
- użytkownik może w tej domenie zmieniać swoje id (np.: root przejść w tryb użytkownika)

- w tej domenie możliwa jest zmiana roli
- w tej domenie proces może zmienić swoje id użytkownika aby edytować jakiś plik (np.: dla programu passwd, musi być domena passwd_t z tym atrybutem, ponieważ proces programu passwd ma id użytkownika, który go wywołał, a żeby nanieść zmiany na plik z hasłami musi mieć id użytkownika)

Dla typów plików kolejno:

- jest to typ pliku
- plik jest dostępny dla administratora
- rodzaj pliku (wykonywalny, tymczasowy etc.)

Warto odnotować, że istnieje możliwość zabronienia rootowi czytania jakiegoś pliku, a więc rzecz niemożliwa w standardowym typie zabezpieczeń, ale de facto taka jest możliwość. Edycja takich plików może być wtedy możliwa przez jakieś specjalne programy ew. wymagające dodatkowej autoryzacji.
Co więcej swojego czasu cytowany na początku guru SELinux, udostępnił w internecie system z hasłem roota. Każdy mógł wejść i robić co chciał. Właśnie, nie wszystko co chciał, mógł np.: wykonać chroot, ale nie mógł czytać niektórych ważnych plików. W efekcie nikt nic nie popsuł i system działał stale bez zarzutu.

Plik users

W tym pliku definiuje się rozpoznawanych użytkowników.

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

Powyższy fragment definiuje użytkownika system_u, który może wykonywać rolę system_r, użytkownika root z odpowiednimi rolami i pojedynczego użytkownika jdoe z jedną rolą. Generalnie w tym pliku można tworzyć wielopoziomowy system autoryzacji, przypisując konkretnym użytkownikom różne role.
Warto zauważyć, że root może mieć dwie różne role. Do przełączania się między rolami służy komenda newrole. Wygodne np.: przy testowaniu czegoś.

Plik users.te

```
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;
allow staff_t unpriv_userdomain:process signal_perms;
dontaudit unpriv_userdomain sysadm_home_dir_t:dir { getattr search };
```

W tym pliku można edytować zachowania domen dla użytkowników, jak i definiować nowe domeny dla uzytkowników.
Pierwsze dwie linijki mówią systemowi, że użytkownik może zmienić typ używanego obiektu. Np.: user jeśli jest w domenie user_t, może zmienić typ urządzenia znakowego z domyślnego dla takiego urządzenia na user_tty_device_t.
W kolejnej linijce określamy zasadę bezwzględnego umożliwiania użytkownikowi w domenie staff_t wysyłania sygnałów do procesów będacyhc w domenach nieuprzywilejowanych (jak user_t).
W ostatniej linijce mamy zasadę niepozwalania nieuprzywilejowanym użytkownikom pobierania atrybutów ani przeszukiwania (czyli zakaz wchodzenia) do katalogów administartora.

Plik users_macros.te

W tym obszernym pliku interesuje nas określanie jakie role mogą mieć domeny
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 };

Powyższe linjki określaką, że dana rola ma do dyspozycji domeny wymienione w klamrach. Pierwsza domena jest domeną domyślną.
Np.: w linijce określającej rolę użytkownika, widzimy że może on znaleźć się w domenie user_netscape_t lub user_irc_t (opisanej wyżej), co oznacza nie mniej nie więcej, że może uruchomić przeglądarkę lub klienta irc. Odpowiednie przejście do danej domeny jest wymuszane przy uruchomieniu któregoś z programów. Takie makra są określone w plikach w podkatalogu macros/program:

Plik ssh_macros.te
Interesuje nas krótki przykład z tego pliku, określający wymuszone przejście z jednej domeny do innej w przypadku uruchomienia programu z pakietu ssh:

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

$1-oznacza parametr
Reguła mówi, że proces z dowolnej domeny, kiedy uruchomi plik wykonywalny programu z pakietu ssh (czyli będzie próbowała wykonać obiekt typu ssh_exec_t), to nie przekaże nowemu procesowi swojej domeny tylko zostanie wymuszone uruchomienie nowego procesu w odpowiedniej domenie dla programów ssh.s

Plik file_context i pliki .fc

Plik file_context jest bardzo ważnym plikiem, przy włączaniu i wyłączaniu SELinux w jądrze, zawiera mianowicie definicje którym plikom nadać jakie typy.
Wymogiem SELinux jest aby pliki miały nadane typy (żeby miały swoje konteksty bezpieczeństwa), system nie zostanie uruchomiony bez spełnienia tego warunku (o tym będzie później).

Przykładowa deklaracja z pliku file_context:

/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

Definujemy tutaj konteksty dla katalogu /home. Korzystając z odpowiednich znaczników, wyróżniamy różne typy dla katalogów, plików i plików z rozszerzeniem .ssh.
Definicje dotyczące kontekstów odpowiednich plików mogą być także spisane w plikach z rozszerzeniem .fc (np.: przy nadawaniu kontekstu dla plików nowo instalowanego programu, lub programu, który możemy kiedyś usuwać i nie chcemy zaśmiecać file_context.

Koniec zabawy, spróbujmy coś zrobić chociaż i tak nie wiemy jeszcze jak

Spróbujmy umożliwić użytkownikowi w domenie user_t używać tcpdump i czytać katalog /etc/selinux: (piszemy to w pliku moja_zmiana.te w podkatalogu /domains)

```
domain_auto_trans(userdomain, netutils_exec_t, netutils_t)
in_user_role(netutils_t)
allow netutils_t user:chr_file rw_file_perms;
r_dir_file(user_t,policy_src_t)
```

Po kolei co zrobiliśmy:

- w pierwszej linijce napisaliśmy, że jeśli proces z dowolnej domeny użytkownika, wywoła plik o typie netutils_exec_t (np.: tcpdump), to automatycznie przechdozimy z domeny użytkownika do domeny netutils_t – bez tego program w ogóle nie zostanie uruchomiony, bo będzie poza jedyną domeną, w której może działać
- następna linijka powoduje, że domena netutils_t jest możliwa do osiągnięcia z dowolnej roli dowolnego użytkownika (a więc odpada nam pisanie masy reguł w w/w plikach, które określają to szczegółowo)
- w trzeciej linijce deklarujemy, że proces w domenia netutils_t będzie mógł pisać i czytać z plików typu chr_file – jest to konieczne aby móc korzystać z terminala
- ostatnia linia deklaruje, że użytkownik w domenie user_t może czytać katalogi i pliki z niego, które są typu policy_src_t – czyli pliki źródła polityki bezpieczeństwa

Ten krótki przykład jak coś wprowadzić, bazuje na kilku makrach. Jak widać znajomość języka m4 naprawdę może ułatwić życie.

Czy zmiany są już widoczne w systemie? Nie, należy jeszcze raz skompilować reguły. Służy do tego program make, z katalogu /etc/selinux. W zależności od parametru:
install –instaluje reguły w systemie ale ich nie ładuje – załadują się po rebootowaniu systemu
load – kompiluje, instaluje i ładuje reguły; nie trzeba uruchamiać ponownie systemu
reload – to samo co wyżej, stymże robi to bezwarunkowo ze wszystkimi regułami. Po wykonaniu make load, system odznacza sobie które reguły są załadowane; jeśli któraś z reguł jest zmieniana, system oznacza, że reguła nie jest załadowana i załaduje ją przy następnym wykonaniu make load, jednak wtedy nei załaduje niezmienionych reguł; reload to obchodzi
relabel – od nowa nadaje konteksty plikom w systemie zgodnie z wytycznymi w pliku file_context i plikach .fc
policy – kompiluje politykę bezpieczeństwa do pliku ale nie instaluje jej, przydatne w testach

Zatem należy jeszcze wykonać polecenie make load i nasze zmiany są wprowadzone. Jak sprawdzić czy wszystko działa? Obserwując komunikaty wyświetlane na ekranie.
Przykładowo komunikat:

```
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
```

oznacza, że proces /bin/login posiadający kontekst bezpieczeństwa wypisany w pierwszej części drugiej linii, próbował zapisywać do obiektu (dokładnie gniazda) o kontekście wypisanym w drugiej części drugiej linii, chociaż nie miał takiego prawa. Oczywiście komunikaty potrafią być mniej skomplikowane....

Jeśli zmieniamy reguły dla jakichś bardzo ważnych programów i w przypadku wystąpienia błędów w konfiguracji reguł nie chcemy, aby taki program nie mógł działać, możemy wykorzystać specjalny program udostępniany przez SELinux avc_toggle, lub podać jądru

parametr „permissive=1”. W ten sposób możemy włączyć tryb, w którym blokowanie dostępu nie jest egzekwowane, lecz pojawiają się wszelkie komunikaty jeśli naruszono reguły. Idealny sposób na testowanie. Avc_toggle służy po prostu do przełączania się między trybami permissive i enforcing.

Czy jesteśmy sami na tej planecie?

Na szczęście istnieją narzędzia, które ułatwiają nam pisanie, rozumienie i testowanie napisanych przez nas reguł. Wyróżniane są dwa narzędzia:

- Tresys Technologies: SE Linux Policy Tools
  Kompleksowe narzędzie do analizy, modyfikowania i ogólnie radzenia sobie z polityką bezpieczeństwa do pobrania z www.tresys.com/selinux (projekt open source)
- audit2allow – prosty program służący do analizowania komentarzy wyrzuconych przez system zabezpieczen o blokowaniu któregos programu, a następnie dopisywaniu odpowiednich reguł aby się to nie powtórzyło.

Dobra, bierzemy go

Skoro taki fajny jest i w ogóle to jak go można zdobyć? Jak już napisałem jest on satndardową częścią jądra 2.6 (odpowiednie opcje w jądrze znajdują się w gałęziach Networking Options i Security Options), zatem dowolny system z tym jądrem już go posiada. Ale nawet wcześniej był on dostępny w dystrybucjach takich jak:

- Fedora Core od wersji 2
- Hardened Gentoo
- Debian
- Suse
- Ubuntu

Co zrobić jęsli chcemy mieć SELinux na wcześniejszum jądrze. Po pierwsze, nie może być to jądra zbyt stare...(pierwsza wersja SELinux była dostosowana do jądra 2.2.12)
Następnie należy ściągnąć (np.: ze stron NSA, nie ogłaszam prostego konkursu na zgadnięcie adresu ich strony) odpowiednie składniki: jądro SELinux (wraz z przykładowymi regułami), łatki na pewne kluczowe programy. Wystarczy wszytsko zainstalować, przekompilować jądra i voila... Pewnie nie będzie chciało od razu działać, toteż czekają takiego śmiałka długie godziny czytania wypluwanych komentarzy i dopisywania reguł.

A teraz, aby powiązać wykład o SELinux z wcześniejszym wykładem, czyli:

Jak SELinux stawia czoło przepełnieniom bufora?

Odpowiedzi można się łatwo domyślić. Konkretnej metody nie ma, bo ogólne metody starczą. Skoro każdy proces jest zamknięty w odpowiedniej domenie, tak aby nie mógł nic namieszać ponad to czym się zajmuje, to przejęcie nad nim kontroli poprzez przepełnienie bufora da intruzowi szansę tylko na działanie w obrębie tej domeny.

Można na to spojrzeć także w ten sposób: SELinux traktuje każdy proces jakby był potencjalnie wadliwy lub wrogi. Temu służy filozofia zamykania procesów zgodnie z ich kontekstami bezpieczeństwa, aby mogły wykonać jak najmniej szkód w systemie, jeśli coś pójdzie nie tak. Zatem SELinux nie przeciwdziała samym takim atakom, nie uniemożliwia przepełnienia bufora, natomiast ogranicza do maksimum możliwe straty. Przywołuje przykład serwera z ogólnodostępnym hasłem roota – nikt go nie popsuł, a pewnie wielu się starało aby choć trochę być znanymi.

Ciekawostka: na forum NSA poświęconego SELinuxowi znalazłem wiadomość, w której jeden z użytkowników napisał, że jest w stanie dla każdego programu, jeśli tylko się boi, że może stać się ofiarą takiego ataku, napisać odpowiednie reguły pilnujące poprawności wprowadzanych danych...

BIBLIOGRAFIA: (LINKOGRAFIA)

http://www.nsa.gov.pl
http://www.crypt.gen.nz/selinux/
http://www.lurking-grue.org/WritingSELinuxPolicyHOWTO.pdf
http://7thguard.net/files/pingwinaria2003.pdf
Wikipedia