STUDIA INFORMATICA 2007
Volume 28 Number 1 (70)

Jacek LACH
Politechnika Śląska, Instytut Informatyki

KONTROLA DOSTĘPU W SYSTEMACH OPERACYJNYCH

Streszczenie. W artykule przedstawiono zwięzłą charakterystykę wybranych metod kontroli dostępu stosowanych w systemach operacyjnych. Krótko przedstawiono również implementację kontroli dostępu w systemie Linux jako przykład praktycznego zastosowania niektórych z opisywanych rozwiązań.

Słowa kluczowe: kontrola dostępu, systemy operacyjne

ACCESS CONTROL IN OPERATING SYSTEMS

Summary. Article contains short characteristics of access control models that are usually used in operating systems. Included is also short presentation of access control implementation in Linux operating system as an example of practical usage of described methods.

Keywords: access control, operating systems

1. Kontrola dostępu

System operacyjny, w ramach którego realizowane jest gromadzenie i przetwarzanie danych, musi zapewnić bezpieczeństwo dokonywanych operacji. Do podstawowych atrybutów bezpieczeństwa danych należy zaliczyć:

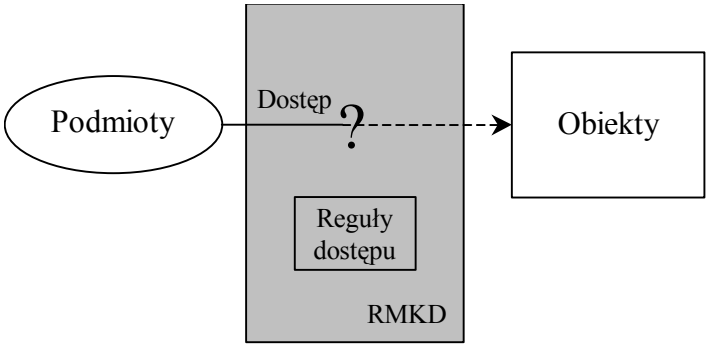
- poufność – zapewnienie, że dane nie mogą być odczytane bez odpowiednich uprawnień,
- integralność – zapewnienie, że dane nie są w nieuprawniony sposób modyfikowane,
- dostępność – umożliwienie wykorzystania danych w dowolnej chwili.

Zapewnieniu bezpieczeństwa danych ma służyć polityka bezpieczeństwa rozumiana jako zbiór ustaleń dotyczących tego, co w systemie jest dozwolone, a co zabronione. Polityka bezpieczeństwa jest egzekwowana z użyciem mechanizmów bezpieczeństwa: metod, procedur

i narzędzi. Jednym z podstawowych mechanizmów, służącym zapewnieniu bezpieczeństwa, jest mechanizm kontroli dostępu w systemie operacyjnym. Mechanizm ten powinien stanowić integralną część systemu operacyjnego (realizacja zabezpieczeń na poziomie aplikacji nie jest wystarczająca) [9]. W dalszej części artykułu scharakteryzowane zostały wybrane modele Kontroli dostępu, stanowiące często podstawę do budowy wspomnianych mechanizmów.

2. Modele kontroli dostępu

Prace nad bezpieczeństwem systemów operacyjnych sięgają lat 70. ubiegłego wieku. Kluczowe rozwiązanie, które zostało zaproponowane, opierało się na wykorzystaniu koncepcji monitora referencyjnego. Schematycznie rozwiązanie zostało przedstawione na rys. 1.



Rys. 1. Koncepcja monitora referencyjnego
Fig. 1. The reference monitor concept

Elementami rozwiązania wykorzystującego monitor referencyjny są:

- Podmioty – aktywne elementy systemu, realizujące przetwarzanie informacji i zmianę stanu obiektów. Podmiot ***s*** jest elementem zbioru podmiotów ***S***.
- Obiekty – pasywne elementy systemu, gromadzące dane. Obiekt ***o*** jest elementem zbioru obiektów ***O***.
- Referencyjny mechanizm kontroli dostępu (RMKD) – podejmuje decyzje o dostępie podmiotów do obiektów na podstawie przypisanych im atrybutów bezpieczeństwa i określonych reguł dostępu. Zakłada się, że mechanizm kontroli dostępu nie może być zmodyfikowany bez odpowiednich uprawnień, nie ma możliwości dostępu do obiektu z pominięciem tego mechanizmu oraz istnieje możliwość zweryfikowania jego poprawności zarówno na poziomie projektu, jak i implementacji.

W systemach operacyjnych często zaimplementowana jest pewna modyfikacja koncepcji monitora referencyjnego. W miejsce referencyjnego mechanizmu kontroli dostępu stosowane są trzy główne rozwiązania:

- uznaniowa kontrola dostępu,
- obowiązkowa kontrola dostępu,
- kontrola dostępu oparta na rolach.

2.1. Uznaniowa kontrola dostępu

Uznaniowa kontrola dostępu (ang. *Discretionary Access Control – DAC*) bazuje na fundamentalnym założeniu, że właściciel obiektu posiada pełną kontrolę nad uprawnieniami do niego. Macierzowy model dla uznaniowej kontroli dostępu został opracowany przez Lampsona [7], a następnie rozbudowany przez Grahama i Denninga [4]. W modelu tym podstawowy element stanowi macierz dostępu, której wiersze indeksowane są podmiotami z *S*, a kolumny obiektami z *O*. Macierz opisuje stan ochrony systemu. Element macierzy (*s, o*) określa prawa dostępu podmiotu *s* do obiektu *o*. Zmiana stanu ochrony systemu może być zrealizowana za pomocą następujących poleceń:

- Create_object() – utworzenie obiektu (dodanie kolumny do macierzy) lub podmiotu (dodanie kolumny i wiersza). Twórca staje się właścicielem utworzonego obiektu lub podmiotu i ma pełne prawa do nadzoru jego uprawnień.
- Delete_object() – usunięcie obiektu lub podmiotu wraz z usunięciem odpowiadających fragmentów macierzy.
- Grant_permission(*r*, *s*, *o*) – nadanie podmiotowi *s* prawa *r* do obiektu *o*. Niemożliwe jest nadanie prawa własności.
- Delete_permission(*r*, *s*, *o*) – odebranie podmiotowi *s* prawa *r* do obiektu *o*.
- Transfer_permission(*r*, *o*, *s*) – przekazanie podmiotowi *s* prawa *r* do obiektu *o*. Operacja może być wykonana, jeżeli podmiot wykonujący polecenie posiada prawo przekazania prawa *r*.
- Read() – odczytanie informacji z macierzy dostępu.

Model ten został następnie rozbudowany przez Harrisona, Ruzzo i Ullmana [5], a jego ścisła definicja umożliwiła dowodzenie własności dotyczących bezpieczeństwa systemów.

Z wykorzystaniem uznaniowej kontroli dostępu wiąże się problem wycieku informacji, wynikający z braku możliwości wpłynięcia właściciela obiektu na prawa przypisywane przez inne podmioty kopiom tego obiektu. Jeżeli użytkownik systemu ma prawo odczytu informacji, może utworzyć jej kopię i udostępnić ją innym użytkownikom niezależnie od pierwotnych ustawień właściciela informacji. Dodatkowo kilku niezaufanych użytkowników może dojść do porozumienia i połączyć swoje uprawnienia. Podstawowe założenie dotyczące peł-

nej kontroli podmiotu nad posiadanymi obiektami wymaga wykorzystania jedynie zaufanej puli podmiotów. Wymaganie to praktycznie jest trudne do spełnienia dla określonej grupy użytkowników. W rzeczywistym systemie dodatkowo należałoby rozróżnić działania realizowane bezpośrednio przez użytkownika od działań podejmowanych na rzecz użytkownika przez uruchomione przez niego oprogramowanie. Więź zaufania należałoby więc w rzeczywistym systemie rozszerzyć dodatkowo o oprogramowanie użytkownika, a tego zrobić nie można (trudno zweryfikować poprawność oprogramowania dostarczonego w wersji binarnej, nie można wykluczyć uruchomienia przez użytkownika „złośliwego" oprogramowania – wirusy czy konie trojańskie). Uznaniowa kontrola dostępu, pomimo swoich wad, jest szeroko stosowana w wielu systemach operacyjnych.

2.2. Obowiązkowa kontrola dostępu

Badania dotyczące bezpieczeństwa doprowadziły do opracowania rozwiązań wykorzystujących obowiązkową kontrolę dostępu, które miały zapewnić ochronę w środowiskach narażonych na istnienie złośliwego oprogramowania. Wiele rozwiązań obowiązkowej kontroli dostępu zostało utworzonych dla rozwiązania problemu bezpieczeństwa wielopoziomowego. Bezpieczeństwo wielopoziomowe (ang. *multilevel security*) polega na przetwarzaniu danych na różnych poziomach wrażliwości. Rozwiązania obowiązkowej kontroli dostępu pochodzą głównie z zastosowań militarnych, gdzie konieczne było kontrolowanie przepływu informacji. Instytucje rządowe mają również ściśle ustaloną hierarchię praw dostępu do informacji, czego odwzorowanie jest wyraźnie widoczne w rozwiązaniach należących do tej grupy. W rozwiązaniach wykorzystujących obowiązkową kontrolę dostępu decyzja o dostępie do obiektu nie jest podejmowana przez użytkownika. Użytkownik może się jedynie podporządkować regułom narzuconym przez wykorzystywany mechanizm kontroli dostępu, nie posiada pełnej kontroli nad danymi, które posiada. Nadrzędnym celem wykorzystania obowiązkowej kontroli dostępu jest zapewnienie, że przepływ danych odbywa się tylko w jednym kierunku.

Modelem odpowiadającym klasyfikacji wojskowej jest model opracowany przez Bella i LaPadulę [1]. Model ten może być uznany za szczególny przypadek modelu kratowego [3]. W modelu tym występuje połączenie obowiązkowej i uznaniowej kontroli dostępu. Podmiot *s* ma uznaniowe prawo odczytu (zapisu) obiektu *o*, jeżeli w macierzy uznaniowej kontroli dostępu dla elementów *s* i *o* zapisano prawo odczytu (zapisu). Jeżeli nie egzekwuje się obowiązkowej kontroli dostępu, to podmiot *s* może czytać (pisać) z obiektu *o*. Wynikiem klasyfikacji (*L*) jest element zbioru klas bezpieczeństwa:

- ściśle tajne (ang. *Top Ssecret* – TS),
- tajne (ang. *Secret* – S),

- poufne (ang. *Confidential* – K),
- niechronione (ang. *Unclassified* – U).

Zbiór klas bezpieczeństwa jest zbiorem uporządkowanym: U < K < S < TS.

Aby sklasyfikować podmioty, używa się określenia poziom uprawnień, aby sklasyfikować obiekty, wykorzystuje się określenie klasyfikacja bezpieczeństwa, $L(s)$ oznacza więc uprawnienia podmiotu s, $L(o)$ oznacza klasyfikację bezpieczeństwa obiektu o. Z każdą klasyfikacją bezpieczeństwa dodatkowo wiąże się zbiór kategorii (C) charakterystycznych dla środowiska, w którym informacje są przetwarzane (np.: NATO, sztab, żandarmeria wojskowa). Poziom bezpieczeństwa definiuje para (L, C). Dla poziomów bezpieczeństwa definiuje się relację dominacji *dom* – poziom bezpieczeństwa (L, C) dominuje poziom bezpieczeństwa (L', C') wtedy i tylko wtedy, gdy $L' \leq L$ i $C' \subseteq C$. Wykorzystanie relacji dominacji umożliwia powiązanie klasyfikacji ze zbiorem kategorii. Jeżeli z klasyfikacji podmiotu wynika dostęp do obiektu, dodatkowo musi być określone prawo dostępu podmiotu do obiektu należącego do określonej kategorii, aby dostęp mógł być rzeczywiście zrealizowany.

W modelu Bella-LaPaduli definiuje się dwie podstawowe własności:

- Warunek bezpieczeństwa prostego – s może czytać o wtedy i tylko wtedy, gdy s *dom* o i s posiada uznaniowe prawo odczytu o. Warunek bezpieczeństwa prostego zapewnia spełnienie własności braku „odczytu w górę", czyli nie jest możliwe odczytanie informacji sklasyfikowanej na wyższym poziomie bezpieczeństwa.
- Własność * – s może zapisać o wtedy i tylko wtedy, gdy o *dom* s i s posiada uznaniowe prawo zapisu o. Własność * określana jest również jako własność uniemożliwiająca „zapis w dół", co uniemożliwia nieautoryzowane przekazanie informacji tajnej na niższy poziom.

Korzystając z powyższych własności, można zdefiniować pojęcie bezpiecznego systemu. Bezpieczny system to taki, w którym spełniony jest warunek bezpieczeństwa prostego i własność *. Podstawowe twierdzenie bezpieczeństwa brzmi: niech Σ będzie systemem z bezpiecznym stanem początkowym σ_0 i niech T będzie zbiorem przekształceń stanu systemu. Jeżeli każdy element zbioru T zachowuje warunek bezpieczeństwa prostego i własność *, to każdy stan $\sigma_i \ \ i > 0$ jest stanem bezpiecznym.

Do podstawowych wad modelu Bella-LaPaduli należy zaliczyć jego słabą adaptowalność do zastosowań w organizacjach wymagających innych rozwiązań niż bezpieczeństwo wielopoziomowe. W modelu tym zakłada się również przekazywanie informacji jedynie za pomocą zwykłych kanałów komunikacyjnych, a nie uwzględnia się możliwości istnienia kanałów ukrytych. Pomimo kontrowersji związanych z modelem Bella-LaPaduli, model ten był pierwszym, ścisłym ujęciem problemu bezpieczeństwa rzeczywistego systemu i przyczyną wielu badań w zakresie kontroli dostępu.

2.2.1. Ukryte kanały

Pojęcie ukrytego kanału przedstawił Lampson w swojej pracy dotyczącej problemu zamknięcia (ang. *confinement problem*) [8]. Ukryte kanały wykorzystują współdzielone zasoby do zrealizowania przesyłu danych. Z punktu widzenia sposobu wykorzystania zasobu do przesyłu informacji za pomocą ukrytego kanału można wyróżnić ich dwa typy:

- Ukryte kanały wykorzystujące składowanie (ang. *covert storage channel*) – do przekazania informacji wykorzystywane są atrybuty współdzielonych zasobów (prawa dostępu, nazwy plików).
- Ukryte kanały wykorzystujące zależności czasowe (ang. *covert timing channel*) – do przekazania informacji wykorzystywane są zależności czasowe, występujące podczas wykorzystania współdzielonych zasobów (kolejność wykorzystania zasobów, czas zajęcia zasobów).

Ukryte kanały dzieli się również ze względu na dostępność kanału dla innych użytkowników systemu na:

- Ukryte kanały ciche (ang. *noiseless covert channels*) – kanały wykorzystujące do komunikacji zasoby dostępne tylko dla nadawcy i odbiorcy.
- Ukryte kanały głośne (ang. *noisy covert channels*) – kanały wykorzystujące do komunikacji zasoby dostępne dla wszystkich.

Do podstawowych własności ukrytego kanału należy zaliczyć istnienie kanału i jego pojemność. Ukryty kanał wymaga współdzielonego zasobu. Wykrywanie ukrytych kanałów jest zagadnieniem trudnym. Do podstawowych metod wykrywania ukrytych kanałów należą:

- Oznaczanie interferencji – jeżeli dwa podmioty mogą ze sobą interferować, to istnieje ukryty kanał.
- Wykorzystanie macierzy współdzielonych zasobów [6] – wiersze macierzy indeksowane są za pomocą atrybutów współdzielonych zasobów, kolumny indeksowane są za pomocą operacji realizujących odczyt bądź modyfikację atrybutów.
- Analiza przepływu informacji.

Pojemnością ukrytego kanału nazywa się maksymalny rozmiar porcji danych, która może być przesłana za jego pomocą. Pojemność wyraża się w bitach. Szybkość transmisji ukrytym kanałem jest ilorazem pojemności kanału i czasu potrzebnego do przesyłu danych.

Eliminacja ukrytych kanałów nie jest prosta i dlatego często dąży się jedynie do ograniczenia możliwości wykorzystania ukrytego kanału, na przykład przez ograniczenie jego pojemności.

2.3. Egzekwowanie typów

Model z egzekwowaniem typów został pierwotnie opisany w [2]. Jest to niskopoziomowy mechanizm obowiązkowej kontroli dostępu, w którym decyzje o dostępie podmiotu do obiektu podejmowane są na podstawie etykiety domeny związanej z podmiotem i etykiety typu związanej z obiektem. W modelu egzekwowania typów każdemu podmiotowi zostaje przypisana domena, a każdemu obiektowi typ. W systemie istnieje zdefiniowana kolekcja operacji dostępu (np. odczyt, zapis, wysłanie sygnału). Dla każdej pary (domena, typ) zdefiniowany jest zbiór dozwolonych operacji, które mogą być wykonane przez procesy przypisane do domeny na obiektach określonego typu. Uprawnienia przechowywane są w bazie danych egzekwowania typów. W bazie, oprócz macierzy określającej dopuszczalne działania dla wymienionej pary (domena, typ), zdefiniowana jest również macierz określająca dopuszczalne działania realizowane między parą podmiotów. Z każdą operacją wykonaną w systemie związany jest jakiś podmiot. Podczas każdej operacji uzyskano dostęp do jakiegoś obiektu. Polityka egzekwowania typów polega na tym, że każdy zrealizowany w systemie dostęp jest zgodny z bazą danych egzekwowania typów. Wśród wad modelu egzekwowania typów jako najpoważniejsze wskazuje się znaczne skomplikowanie konfiguracji kontroli dostępu, wynikające z liczby wchodzących w interakcje podmiotów i obiektów, oraz mało naturalne odwzorowanie standardowej struktury systemu na tabelaryczną strukturę modelu.

2.4. Kontrola dostępu z wykorzystaniem ról

Kontrola dostępu z wykorzystaniem ról używa zamiast uprawnień przypisywanych konkretnym podmiotom uprawnienia przypisywane do ról, które użytkownicy mogą pełnić w systemie. Rola jest tu rozumiana jako stanowisko pracy ze ściśle określonymi zadaniami oraz uprawnieniami niezbędnymi do wykonania tych zadań. Najpierw definiowany jest zbiór ról, następnie rolom przydzielane są określone uprawnienia do obiektów, a na końcu użytkownikom przypisywane są odpowiednie role. Ważne jest w tym przypadku wiązanie uprawnień przypisanych do określonej roli z użytkownikiem, a nie podmiotem (czyli na przykład procesem użytkownika). Model ten jest odpowiedni dla większości firm, gdzie uprawnienia nie są uzależnione od tego kim jest dana osoba, a jedynie od tego jakie jest jej stanowisko. Dzięki takiemu podejściu możliwe jest utrzymanie tej samej polityki dostępu dla zmieniającej się obsady osobowej – osobie przy zmianie stanowiska przypisywana jest po prostu nowa rola. W ogólnym modelu definiuje się wykorzystanie zbioru użytkowników, zbioru uprawnień dostępu, zbioru ról oraz zbioru sesji. Sesja jest inicjowana przez użytkownika. W ramach sesji wykorzystuje się określone role, w których użytkownik może występować.

Kontrola dostępu z wykorzystaniem ról jest obojętna na stosowaną politykę. Jest raczej sposobem na zapis polityki, niż implementacją wybranej polityki. Polityka, będąca efektem

zastosowania tego modelu, jest wynikiem szczegółowej konfiguracji i współdziałania elementów modelu na zasadach zdefiniowanych przez administratora. Niewątpliwą zaletą modelu jest możliwość ewolucji polityki kontroli dostępu wraz ze zmieniającymi się potrzebami organizacji. Jako wadę modelu wskazuje się potrzebę wykorzystania dodatkowo innych rozwiązań (np. egzekwowania typów) w celu utworzenia bezpiecznego systemu.

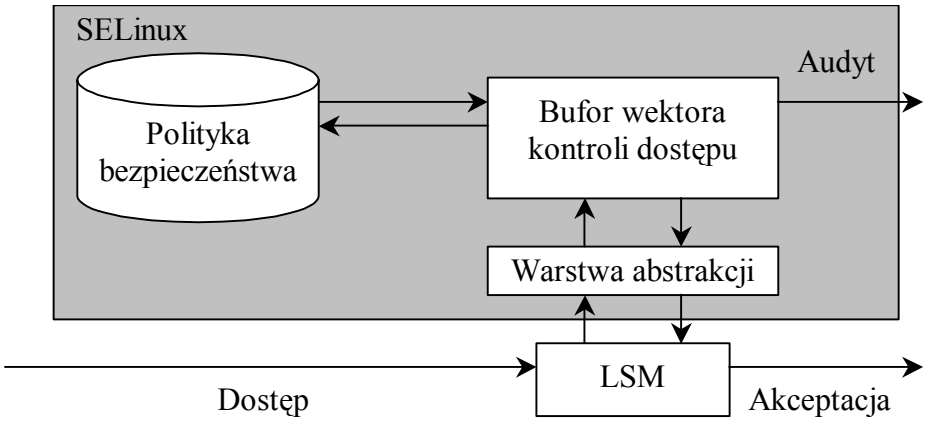
3. Implementacja kontroli dostępu w systemie Linux

W tradycyjnym modelu obowiązkowej kontroli dostępu występuje silny związek z bezpieczeństwem wielopoziomowym – decyzje o dostępie podejmowane są na podstawie klasyfikacji obiektu i uprawnień podmiotu. Takie podejście wprowadza zbyt duże ograniczenia dotyczące możliwości wykorzystania wspomnianego modelu dla wielu środowisk o mniej ścisłej hierarchii niż wojskowa. Problemami są: zapewnienie integralności danych, separacja odpowiedzialności oraz wymagania dotyczące minimalnych uprawnień. Te ograniczenia tradycyjnego modelu były przyczyną utworzenia ogólnej architektury kontroli dostępu Flask [10]. Flask dostarcza elastycznego wsparcia dla polityk bezpieczeństwa, umożliwiając wykorzystanie obowiązkowej kontroli dostępu w sposób odpowiedni dla postawionych wymagań bezpieczeństwa. Jest to możliwe dzięki oddzieleniu części decyzyjnej polityki bezpieczeństwa od mechanizmu egzekwowania polityki. Część decyzyjna zawarta jest w serwerze bezpieczeństwa. Egzekwowaniem polityki zajmuje się menadżer bezpieczeństwa. W architekturze Flask każdy obiekt i proces ma własny kontekst bezpieczeństwa. Podczas realizacji dostępu menadżer bezpieczeństwa przekazuje do serwera bezpieczeństwa parę identyfikatorów (procesu i obiektu), które są odwzorowywane na kontekst bezpieczeństwa i na tej podstawie podejmuje decyzję o możliwości zrealizowania operacji. Wyniki podejmowanych decyzji są zapamiętywane w pamięci podręcznej, określanej jako bufor wektora dostępu (ang. *Access Vector Cache*).

Security Enhanced Linux jest rozszerzeniem tradycyjnej, uznaniowej kontroli dostępu, wykorzystywanej w systemie operacyjnym Linux. SELinux jest implementacją architektury Flask dla systemu Linux. Implementacja tego rozszerzenia pociągnęła za sobą utworzenie w ramach jądra systemu operacyjnego modułów bezpieczeństwa (ang. *Linux Security Modules*) [11]. Moduły bezpieczeństwa to prosta, ogólna architektura kontroli dostępu, obecna w standardowej wersji jądra systemu operacyjnego, umożliwiająca wykorzystanie różnych modeli kontroli dostępu zaimplementowanych w postaci ładowalnych modułów jądra.

SELinux umożliwia wykorzystanie obowiązkowej kontroli dostępu przez użycie rozwiązań zaproponowanych w modelu z egzekwowaniem typów, modelu kontroli dostępu na podstawie ról oraz modelu bezpieczeństwa wielopoziomowego. Kontekst bezpieczeństwa ma

trzy atrybuty: tożsamość, rolę i typ. Każdy proces w systemie ma swoją tożsamość (odrębną od identyfikatora użytkownika systemowego), determinującą zbiór ról, a tym samym domen, w ramach których może występować, określając w ten sposób dostęp do obiektów systemu. Każdemu obiektowi w systemie przypisany jest typ. Wykorzystanie egzekwowania typów ma zapewnić bardzo szczegółową konfigurację kontroli dostępu. Wykorzystanie kontroli dostępu na podstawie ról umożliwia użytkownikowi przejścia pomiędzy domenami oraz ułatwia tworzenie polityki bezpieczeństwa dla systemu. Uproszczony diagram architektury SELinuxa przedstawiony został na rys. 2.



Rys. 2. Uproszczony diagram architektury SELinuxa
Fig. 2. Simplified diagram of SELinux architecture

Konfiguracja polityki bezpieczeństwa jest realizowana z wykorzystaniem dedykowanego języka i zawiera wyrażenia egzekwowania typów (deklaracje atrybutów, deklaracje typów, reguły zmiany typów) oraz wyrażenia kontroli dostępu z wykorzystaniem ról (deklaracje ról, definicje dominacji ról). Konfiguracja polityki bezpieczeństwa jest aktualnie jednym z większych wyzwań stojących przed administratorem systemu oraz celem prac kilku projektów zmierzających do utworzenia narzędzi ułatwiających wykorzystanie bardzo silnego mechanizmu bezpieczeństwa jakim jest SELinux.

LITERATURA

1. Bell D., LaPadula L.: Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997 Rev. 1, MITRE Corporation, Bedford 1975.
2. Boebert W., Kain R.: A Practical Alternative to Hierarchical Integrity Policies. Proceedings of the 8th National Computer Security Conference, 1985, s. 18÷27.
3. Denning D. E.: Alattice model of secure information flow. Communications of the ACM, 1981, 24(8), s. 533÷536.

4. Graham G., Denning P.: Protection: principles and practices. Proceedings of the AFIPS Spring Joint Computer Conference, 1972, s. 417÷429.
5. Harrison M., Ruzzo W., Ullman J.: Protection in operating systems. Communications of the ACM, 1976, 19(8), s. 461÷471
6. Kemerer R.: Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels. ACM Transactions on Computer Systems, 1983, 1(3), s. 256÷277.
7. Lampson B.: Protection. Proceedings of the 5th Princeton Conference on Information and System Sciences, 1971, s. 437÷443.
8. Lampson B.: A Note on the Confinement Problem. Communications of the ACM, 1976, 16(10), s. 613÷615.
9. Loscocco P., Smalley S., Muckelbauer P., Taylor R., Turner S., Farrell J.: The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. Proceedings of the 21st National Information Systems Security Conference, 1998, s. 303÷314.
10. Spencer R., Smalley S., Loscocco P., Hibler M., Andersen D., Lepreau J.: The Flask Security Architecture: System Support for Diverse Security Policies. Proceedings of the Eighth USENIX Security Symposium, 1999, s. 123÷139.
11. Wright C., Cowan C., Morris J., Smalley S., KroahHartman G.: Linux security modules: General security support for the Linux kernel. Proceedings of the USENIX Security Symposium, 2002.

Recenzent: Dr inż. Krzysztof Nałęcki

Wpłynęło do Redakcji 2 grudnia 2006 r.

Abstract

Operating system must ensure secure environment for storing and processing data. Important role in building secure environment plays security policy which can be enforced by access control model implemented in the system. Traditional model available in UNIX operating system is called discretionary access control. Owners use their discretion to decide whether access to objects they own should be granted. The problem is that owner has no control over copies of data other users may have done. This way data can be spread in a way that was not meant by the owner. Because of the problems with DAC much work has been done on mandatory access control, where the access control decisions were not at the discretion of individual users or administrator. Example of mandatory access control is Bell-LaPadula

model. Mandatory access control is not well suited for organizations that do not comply with multilevel security, also traditional models do not cope well with covert channels. Type enforcement provides a means for flexible mandatory access control which can be adapted for different security goals. Role based access control is another solution for creating secure access control mechanisms though as such it does not imply using any in particular. Implementation of SELinux is example of using combination of type enforcement and role based access control to build flexible yet fine-grained mandatory access control. This article presents short overview of common access control models and shortly describes SELinux.

Adres

Jacek LACH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, jacek.lach@polsl.pl .