Projekt TrustedBSD jako klucz do bezpieczeństwa systemu FreeBSD

Wojciech A. Koszek [1]

dunstan@FreeBSD.czest.pl

[1] IX Liceum Ogólnokształcące im. C. K. Norwida w Częstochowie
klasa II, profil politechniczny

Streszczenie

Wymogi stawiane bezpieczeństwu systemów operacyjnych wciąż rosną: konieczna staje się kontrola dostępu do zasobów maszyny już na poziomie jądra systemu, wymagane są efektywne mechanizmy logowania i wykrywania anomaliów. Wychodząc naprzeciw tym zapotrzebowaniom, kilka lat temu rozpoczęto prace nad TrustedBSD [1]. Głównym celem projektu TrustedBSD jest stworzenie implementacji machanizmów bezpieczeństwa zgodnych ze standardem Posix.1e [2] przeznaczonych do włączenia w kod źródłowy systemu FreeBSD [3]. Obecny stan prac umożliwił integrację cześci już stworzonego kodu z systemem bazowym, czego efektem jest bardzo rozbudowany podsystem bezpieczeństwa obecny we FreeBSD od wersji 5-tej systemu. Niniejsza praca dotyczy zalet związanych z wykorzystaniem nowoczesnych mechanizmów bezpieczeństwa we FreeBSD, opisu ich możliwości i sposobów konfiguracji. Jako przykład wykorzystany zostanie prosty mechanizm TPE zaimplementowany przez autora.

1 Wprowadzenie

Obecne machanizmy bezpieczeństwa dostarczane przez systemy operacyjne możemy podzielić na mechanizmy działające w przestrzeni jądra bądź przestrzeni użytkownika. Od dawna wiadomo, że te ostatnie posiadały liczne słabe punkty znane już w w fazie projektowania (problem zmiennych *LD_* linkera, problemy znacznika *'nosuid'* partycji, biblioteki wprowadzające dodatkowy stopień bezpieczeństwa poprzez zastępowanie funkcji biblioteki *libc* alternatywnymi zamiennikami). Praktyka stała się jedynie potwierdzeniem tej tezy co sprawiło, iż narzędzia te postrzegane są bardziej jako środki prewencyjne niż dostarczające wysokiego poziomu prawdziwego bezpieczeństwa. Wprowadzenie dodatkowej kontroli w przestrzeni jądra wynika z niezwykle prostego faktu: jądro jest unikalnym procesem, posiadającym dostęp do każdego, fizycznego zasobu działającej maszyny. Jednymi z głównych funkcji jądra są:

> rzetelny podział dostępnej pamięci wolnodostępnej między wiele współdziałających procesów poprzez podsystem pamięci wirtualnej.

wykorzystanie modularnych mechanizmów, dzięki którym możliwy staje się dostęp do danych dyskowych w różnorakich formatach (wiele dysków, partycji, systemów plików)

zapewnienie rzetelnej sygnalizacji o zaistnieniu pewnej sytuacji

wprowadzenie sposobów wymiany danych między dostępnymi procesami (w tym możliwość komunikacji między danymi jądra i użytkownika).

zapewnienie dostępu do zasobów maszyny (rejestry/instrukcje CPU, dostęp do systemu BIOS, portów wejścia/wyjścia)

Poprzez brak większości organiczeń spotykanych w narzędziach bezpieczeństwa przestrzeni użytkownika, jądro staje się doskonałym miejscem na kontrolowanie wszystkich wymienionych zasobów. Choć nie wszystkie wymienione funkcje jądra zaprojektowane były z myślą o bezpieczeństwie, spotykane dziś systemy muszą posiadać możliwość kontroli każdej z nich, co ze względów złożoności i odmienności oprogramowania, staje się zadaniem bardzo trudnym. Należy pamiętać o tym, iż dla nowoczesnych systemów typu UNIX normalną sytuacją jest praca na maszynie z dużą ilością aktywnych użytkowników, między których dzielone są ograniczone zasoby systemowe oraz którzy mogą wykonywać wiele niezależnych zadań. Rozwój technologiczny sprawił, iż do wszystkich wymienionych funkcji dołączyć trzeba również możliwość pracy na maszynach wieloprocesorowych i możliwość współdzielenia określonych zasobów poprzez sieć komputerową. Oprócz wspomnianych problemów, istotną trudnością w realizacji tak rozbudowanego mechanizmu bezpieczeństwa staje się różnica między przestrzenią jądra i użytkownika w sposobie reprezentacji obiektów systemowych (pliki, procesy, sposoby dostępu do pamięci). Przykładem niech będzie sposób reprezentacji otwartego pliku, który w przestrzeni użytkownika widoczny jest jako unikalny deskryptor [4]. W przestrzeni jądra plik postrzegany jest jako wirualny węzeł (*ang. vnode*) w danym systemie plików ([5],[6],[7]). Choć obiekt pozostaje ten sam, interfejs używany do dostępu różni się diametralnie.

Mimo ogromnego nakładu pracy koniecznego do opracowania zabezpieczeń spełniających wyżej wymionione założenia, naukowcom oraz ekspertom od bezpieczeństwa udało się stworzyć odpowiedni model, oraz określić wymogi stawiane zaproponowanej funkcjonalności. Efektem ich badań i doświadczeń stał się standard Posix.1e [2]. Standard opisuje nie tylko wymaganą funkcjonalność lecz przede wszystkim interfejs programistyczny oraz interfejs użytkownika. Dzięki temu, sposób działania stworzonych narzędzi powinien być w znacznej mierze przenośny między systemami zgodnymi z Posix.1e.

2 Posix.1e

Do głównych założeń stawianych przez standard Posix.1e należą:

1. listy kontroli dostępu (*ang. ACL - Access Control Lists*)
2. audytowanie zdarzeń (*ang. Auditing*)
3. separacja przywilejów (*ang. Capabilities - separation of privilege*)
4. obowiązkowa kontrola dostępu (*ang. MAC - Mandatory Access Control*)
5. możliwość oznaczania obiektów (*ang. Labeling*)

2.1 Listy kontroli dostepu

Standardowe mechanizmy kontroli dostępu do plików spotykane we wszystkich wariantach systemu UNIX bazują na możliwości oznaczenia pliku jako przeznaczonego do odczytu, zapisu bądź wykonania dla właściciela pliku, grupy posiadającej plik bądź innych użytkowników [8]. Niestety w bardziej złożonych przypadkach mechanizm ten może okazać się niewystarczający – chcąc współdzielić zasoby dyskowe między dwóch użytkowników, administrator musi stworzyć osobną grupę, do której musi kolejno przypisać każdego z nich. W przypadku większej liczby użytkowników metoda staje się nieefektywna. Listy kontroli dostępu dają możliwość udostępnienia pliku dla wyselekcjonowanych użytkowników i grup.

2.2 Audytowanie zdarzeń

Audytowanie to możliwość rejestrowania informacji dotyczących zdarzeń mogących mieć wpływ na bezpieczeństwo. Prace nad wprowadzeniem tej funkcjonalności trwają. Projekt audytowania w jądrze FreeBSD posiada swoją stronę internetową [9], która w najbliższym czasie zostanie uzupełniona o szczegóły wprowadzanych zmian oraz sposobu implementacji.

2.3 Separacja przywilejów

Największym problemem związanym z bezpieczeństwem programów systemowych jest konieczność uruchamiania niektórych aplikacji z przywilejami administratora. Dzieje się tak z racji określonej funkcjonalności wymaganej do poprawnego działania programu (dostęp do zabezpieczonych plików, chronionych obszarów pamięci). Jako, iż w obecnych systemach rodziny UNIX uprawnienia administratora umożliwiają dokonanie ogromnej szkody, postanowiono podzielić je na większy zestaw mniejszych przywilejów niezbędnych do wykonania danego zadania (przykładowo, aplikacja wiążąca port o numerze niższym niż 1024 z gniazdem sieciowym nie musi posiadać dostępu do pamięci jądra, lecz tylko możliwość wywołania funkcji *bind(2)*).

2.4 Obowiązkowa kontrola dostępu

Jednym ze sposobów komunikacji z przestrzenią użytkownika są wywołania systemowe, czyli funkcje w przestrzeni jądra realizujące podstawową funkcjonalność (*open(2)*, *close(2)*, *socket(2)*). Przechwytywanie określonych wywołań systemowych i przeprowadzanie testów jest jedną z możliwych dróg uzyskania podwyższonego stopnia kontroli i było wykorzystywane już wielokrotnie w historii Wolnodostępnych systemów operacyjnych.

Warstwa MAC umożliwia dokonywanie testu na argumentach przekazanych do funkcji systemowej. Możliwie jest również kontrolowanie poszczególnych etapów działania funkcji, jeżeli zachodzi taka potrzeba (*execve(2)*). Podejmowanie odpowiednich, wcześniej ustalonych decyzji odbywa się na podstawie wyniku testu. Kontroli można dokonywać globalnie, w obrębie systemu bądź lokalnie, w obrębie procesów i plików posiadających tą samą etykietę.

2.5 Możliwość flagowania obiektów

Możliwość flagowania obiektów dodatkowymi danymi to funkcjonalność, która nie jest związana bezpośrednio z bezpieczeństwem. Dzięki niej system plików oraz struktury reprezentujące procesy w systemie mogą zostać oflagowane danymi niosącymi potrzebne informacje (etykieta, para nazwa/wartość), przez co kontrola przepływu bądź badanie wzajemnych relacji w systemie staje się o wiele łatwiejsze.

3 TrustedBSD

TrustedBSD jest odpowiedzią na brak istniejącej implementacji mechanizmów zgodnych z Posix.1e w systemach BSD (*ang. Berkley Software Distribution*). Choć głównym celem projektu jest uzyskanie zgodności implementacji w systemie FreeBSD z Posix.1e, spora część funkcjonalności obecnie tworzonego kodu powinna dać się przenieść na inne systemy rodziny BSD [10].

W chwili pisania tego artykułu integracji z FreeBSD uległy mechanizmy MAC oraz ACL [11]. Mechanizm separacji przywilejów jest dostępny poprzez repozytorium projektu TrustedBSD. Mechanizm audytowania zaprezentowany na konferencji BSDCan2005 [12] nie jest jeszcze publicznie dostępny, stąd też autor nie jest w stanie dostarczyć bardziej szczegółowych informacji na temat funkcjonalności.

3.1 Implementacja

Opisując projekt TrustedBSD nie sposób pominąć kilku szczegółów implementacji. Praktycznie wszystkie wymagania stawiane przez Posix.1e wiążą się z wprowadzeniem zmian do istniejących struktur danych. Istnieje konieczność powiązania dodatkowych informacji z wewnętrznymi strukturami reprezentującymi obiekty w jądrze. Stąd też dokonano modyfikacji systemu plików oraz struktur procesów.

Mechanizm list kontroli dostępu w systemie plików UFS2 bazuje na funkcjonalności rozszerzonych atrybutów. Istnieją dwie przestrzenie adresowe - przestrzeń adresowa użytkownika (`EXTATTR_NAMESPACE_USER`) i systemowa (`EXTATTR_NAMESPACE_SYSTEM`). Pierwsza przenaczona jest do modyfikacji przez zwykłych użytkowników, z kolei druga, jedynie przez administratora. Poprzez przypisanie do poszczególnych plików pary w postaci nazwy zmiennej i jej wartości, użytkownik i administrator mogą uzyskać dowiązanie dowolnych danych do numeru węzła pliku nie naruszając głównej zawartości. Dzięki rozszerzonym atrybutom, możliwe staje się również załączenie informacji o liście kontroli, co wykorzystano w obecnej implementacji.

Mechanizm obowiązkowej kontroli dostępu został zaimplementowany poprzez wprowadzenie dodatkowych funkcji przechwytujących argumenty funkcji w jądrze systemu, które mogą stać się potencjalnymi kandydatami do kontroli podczas tworzenia polityki bezpieczeństwa. Procesy, pliki i inne struktury można oznaczyć jednakowymi flagami, przez co mogą być postrzegane jako jednolita polityka bezpieczeństwa. Dana polityka może dotyczyć konkretnej cześć systemu, bądź też kilku. Możliwe jest oczywiście łączenie wielu polityk, tworząc kompleksowy system zaawansowanej kontroli. Jeżeli administrator zdecydował na włączenie mechanizmu MAC w trakcie kompilacji jądra, podczas działania systemu normalna struktura wywołania systemowego ulega zmianie. Tuż po wywołaniu danej funkcji, kontrola przekazywana jest do mechanizmu MAC. W zależności

od rodzaju funkcji, odnajdywana jest odpowiednia funkcja kontrolna, która jest w stanie dokonać porządanych testów bezpieczeństwa. Od rezultatu testów zależy dalszy sposób postępowania - w przypadku powodzenia normalny przebieg wykonywania jest przywracany. W przypadku niespełnienia określonych warunków, wywołanie systemowe poprzez funkcję kontrolną zwraca błąd *EPERM* do aplikacji użytkownika.

Cały szkielet MAC działa na wielu poziomach interpretacji wewnętrznych struktur danych systemu - od udostępniania argumentów kluczowych z punktu bezpieczeństwa funkcji (*socket(2)*, *pipe(2)*, *execve(2)*), poprzez kontrolę sieciowych buforów pamięci *mbuf(9)* (*ang. memory buffer*), aż po mechanizmy komunikacji znane z System V (*semctl(2)*, *semget(2)*, *semop(2)*) Warto nadmienić, iż w przypadku braku funkcji kontrolnej, sposób przepływu instrukcji w wywołaniu systemowym nie ulega modyfikacjom. Rozwiązanie MAC jest szczególnie ciekawe, gdyż istnieją już moduły wykorzystujące cechy mechanizmu.

Z punktu widzenia administratora, podobnie jak większość podsystemów jądra FreeBSD, tak i mechanizm MAC jest modularny i rozszerzalny. O ile główne źródła konieczne do uruchomienia systemu muszą być skompilowane statycznie, o tyle większość modułów wprowadzających konkretne polityki bezpieczeństwa może być ładowana w trakcie działania systemu.

3.2 Korzyści

Autor pracy wykorzystał możliwości KLD (*ang. Dynamic Kernel Linker*) w jądrze systemu FreeBSD do stworzenia prostego mechanizmu TPE (*ang. Trusted Path Execution*), wykorzystując przy tym możliwości jakie daje przechwytywane wywołań systemowych. Dzięki TPE przed każdym uruchomieniem pliku wykonywalnego sprawdzana była jego suma kontrolna pliku, z możliwością logowania poprzez standardowy mechanizm *syslog(3)*. Do kalkulacji sumy kontrolnej zostały wykorzystane funkcje SHA1 oraz MD5. Mimo uzyskania wystarczającej funkcjonalności, istnieje wiele problemów, z którymi mogą spotkać się użytkownicy tego typu oprogramowania:

> przechwytywanie nie jest operacją przenaczoną do kontroli argumentów. Istnieje potrzeba wprowadzenia kontroli w głównym ciele funkcji, gdyż daje to pewność, że test zostanie wykonany. Mechanizm MAC włączany jest poprzez dyrektywy `#define` do kodu źródłowego. Ewentualna próba obejścia mechanizmu polega na uruchomieniu zmodyfikowanego jądra bez skompilowanego zabezpieczenia bądź jądra z podmienioną funkcją.
>
> wady implementacyjne mogą w znacznym stopniu zmiejszyć stopień bezpieczeństwa. Dzięki MAC programista jest w stanie zmiejszyć ilość kodu do niezbędnego minimum dokonując jedynie pożądanych testów.
>
> brak rozszerzalności i elastyczności. Mimo, iż kod znajduje się w przestrzeni jądra, nie ma możliwości zaawansowanego selekcjonowania sytuacji, w których kalkulacja ma mieć miejsce.

Tuż po ukończeniu projektu autora opublikowany został kod o bardziej rozbudowanej funkcjonalności [13] stworzony przez jednego z programistów systemu FreeBSD (*Christian S.J. Peron*) bazujący na mechaniźmie MAC. Dzięki istniejącej funkcjonalności, możliwe stało się kontrolowanie nie tylko funkcji *execve(2)*, ale również funkcji operujących na węzłach wirtualnych (*ang. vnodes*). Propozycje zmian zostały zgłoszone do

osoby rozwijającej moduł, zaś znaleziony błąd w narzędziu przestrzeni użytkownika naprawiony.

3.3 Narzędzia systemowe i konfiguracja

Konfiguracja tak złożonej funkcjonalności byłaby niezwykle trudna bez wsparcia ze strony narzędzi przestrzeni użytkownika i samego systemu FreeBSD. Dlatego też w źródłach systemu dostępne są już narzędzia oraz biblioteki umożliwiające integrację oprogramowania z rozszerzeniami bezpieczeństwa. Konfiguracja ACL polega na włączeniu przy pomocy narzędzia *tunefs(8)* globalnej flagi w systemie plików, w którym listy kontroli dostępu mają być wykorzystywane. Dalsza konfiguracja opiera się na przypisywaniu poszczególnych list do plików. Możliwe jest to dzięki narzędziu *setfacl(1)*, zaś pobieranie obecnie skonfigurowanych list dzięki narzędziu *getfacl(1)*. Standardowe uprawnienia są konwertowane przez *getfacl(1)* na kształt skonfigurowanej listy dostępu:

```
$ getfacl /etc/mail/sendmail.cf
#file:/etc/mail/sendmail.cf
#owner:0
#group:0
user::rw-
group::r--
other::r--
```

Przypisanie dodatkowych uprawnień odbywa się poprzez polecenia w podobnym formacie:

```
# setfacl -m u:dunstan:x,g:sadm-mail:rwx /etc/mail/sendmail.cf
$ getfacl /etc/mail/sendmail.cf
#file:/etc/mail/sendmail.cf
#owner:0
#group:0
user::rw-
user:dunstan:--x
group::r--
group:sadm-mail:rwx
mask::rwx
other::r--
```

Jak widać na zamieszczonym przykładzie, możliwe staje się przypisanie nawet arbitralnych wartości liście kontroli dostępu. Poprzez odpowiednie użycie narzędzi *getfacl(1)* i *setfacl(1)* można łatwo przenosić zestawy uprawnień, co ułatwia administrację.

Wykorzystanie warstwy MAC staje się bardziej złożone. Istnieje kilka modułów, implementujących prostą funkcjonalność:

> *mac_seeotheruids(4)* - polityka wprowadza globalną kontrolę w warstwie procesów uniemożliwiając użytkownikowi oglądanie procesów do niego nie należących. Po załadowaniu modułu z polityką, jest ona aktywna i praktycznie nie wymaga konfiguracji.

mac_partition(4) - polityka działająca w warstwie procesów. Umożliwia tworzenie "partycji" procesów, przez co są one w stanie widzieć jedynie procesy z podobną etykietą. Aplikacjami wykorzystywanymi do testowania i budowy narzędzi bazujących na MAC są *setfmac(8)* oraz *setpmac(8)*. Dzięki nim możliwe jest przypisywanie określonej etykiety wymaganej przez politykę MAC do plików i procesów. Przykładowo, uruchomienie powłoki '*/bin/csh*' z etykietą 'partition/15' wygląda następująco:

```
# setpmac 'partition/15' /bin/csh
#ps axuw
USER   PID %CPU %MEM   VSZ   RSS  TT  STAT STARTED      TIME COMMAND
root  1828  1,0  0,4  2524  1908  p4  S    16:46     0:00,02 /bin/csh
root  1829  0,0  0,2  1580   952  p4  R+   16:46     0:00,00 ps axuw
```

Jak widać, następuje ograniczenie widoczności do procesu powłoki.

mac_portacl(4) - polityka umożliwiająca szczegółową kontrolę nad argumentami przekazanymi do funkcji *bind(2)*. Poprzez dołączenie listy kontroli dostępu, administrator jest w stanie ograniczać możliwość przywiązywania portów sieciowych przez proces. Lista kontroli przyjmuje postać:

```
typ:id:protocol:port[,idtype:id:protocol:port,...]
```

gdzie `typ` może przyjąć wartość "*gid*" dla identyfikatorów grupy bądź "*uid*" dla oznaczania identyfikatorów użytkownika, `id` jest numeryczną wartością identyfikatora, `protocol` to typ protokołu – "*tcp*" bądź "*udp*", a `port` to numer portu ograniczanego.

mac_bsdextended(4) - polityka wprowadza rozszerzenia do standardowego modelu zabezpieczeń bazującego na identyfikatorach użytkownika i grupy. Dzięki kompleksowym opcjom konfiguracji istnieje możliwość wprowadzenia wzajemnych ograniczeń dotyczących widoczności plików i procesów w systemie na podstawie identyfikatorów użytkownika i grupy. Więcej informacji dotyczących konfiguracji polityki *mac_bsdextended(3)* znajduje się na stronie podręcznika systemowego narzędzia *ugidfw(8)*. Podczas pisania tego artykułu autor poprawił błąd w tym narzędziu, a zaproponowana poprawka została włączona do źródeł systemu.

Oprócz opisanych modułów, we FreeBSD znajdują się również inne polityki, w których kontrola polega na sposobie przepływu informacji: poprzez wprowadzenie hierarchii oznaczenia procesów i plików, możliwa staje się kontrola na podstawie priorytetu. Do polityk tych zaliczamy *mac_biba(4)* (*BIBA data integrity policy*), *mac_mls(4)* (*Multi-Level Security policy*) oraz *mac_lomac(4)* (*Low-Watermark policy*). Ze względu na złożoność i różnice między politykami, artykuł ten nie porusza tej tematyki. Osoby zainteresowane znajdą więcej informacji na stronach *man(1)* poświęconych modułom i narzędziom [14]

4 Podsumowanie

Autor zaprezentował możliwości nowoczesnych mechanizmów bezpieczeństwa w systemie FreeBSD. Sam system sprawdza się doskonale w środowiskach produkcyjnych [15]

i naukowych [16], [17], stąd też tak rozbudowane podsystemy kontroli stają się koniecznością. Pozostaje mieć nadzieję, że organizacje wspierające rozwój opisanych rozwiązań ([18], [19], [20]) poprzez dotacje umożliwią kontynuację prac i w ostateczności, implementację wszystkich mechanizmów zgodnych z Posix.1e w systemie FreeBSD.

Literatura

[1] *www.TrustedBSD.org*, strona projektu TrustedBSD

[2] *http://wt.xpilot.org/publications/posix.1e/*, standard Posix.1e

[3] *http://www.FreeBSD.org*, strona projektu FreeBSD

[4] "Advanced Programming in the UNIX environment", W. Richard Stevens

[5] "The Design and Implementation of the 4.4BSD Operating System", Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman

[6] "UNIX internals - The New Frontiers", Uresh Vahalia

[7] "The Design and Implementation of the FreeBSD Operating System", Marshall Kirk McKusick, George Neville-Neil.

[8] *chmod(1)*, strona opisująca standardowy zestaw uprawnień

[9] *http://www.OpenBSM.org*, projekt audytowania w systemie FreeBSD

[10] *http://openbsd.cz/~milos/mac/*, "NetBSD port of the TrustedBSD MAC Framework"

[11] katalogi *u*fs, *s*ecurity oraz *k*ern w hierarchii *src/sys*.

[12] *http://www.bsdcan.org/2005/*, strona konferencji BSDCan2005

[13] *http://people.freebsd.org/~csjp/mac/*, moduł TPE o nazwie *mac_chkexec*

[14] strony podręcznika systemowego: *mac(3)*, *mac(9)*, *setfmac(8)*, *setpmac(8)*,*acl(3)*, *setfacl(1)*.

[15] *http://www.offmyserver.com/cgi-bin/store/cluster.html*, The TechTV Screen Savers Cluster

[16] *http://people.freebsd.org/~brooks/papers/bsdcon2003/*,
"Building a High-performance Computing Cluster Using FreeBSD"

[17] *http://people.freebsd.org/~brooks/papers/usebsd2004/*, "Grid Computing with FreeBSD"

[18] *http://www.darpa.mil/*, Defense Advanced Research Projects Agency (DARPA)

[19] *http://www.nai.com*, Network Associates

[20] *http://www.nsa.gov*, National Security Agency