BEZPIECZEŃSTWO SYSTEMU Z PAX

• Błędy oprogramowania

Podstawową przyczyną, dla której możliwe są ataki na systemy komputerowe,są błędy w oprogramowaniu. Do najpoważniejszych należy błąd przepełnienia bufora (ang. buffer overflow) - oraz pokrewne - pozwalające na wprowadzenie do procesu obcego kodu. Niestety nie możemy liczyć na poprawienie raz na zawsze wszystkich napisanych programów, więc dobre systemy operacyjne powinny udostępniać mechanizmy zmniejszające negatywne skutki ataku z wykorzystaniem błędów programistycznych.

• Czym jest PaX?

PaX to łata dla jądra Linuksa zapewniająca większe bezpieczeństwo systemu i ochronę pamięci. Autorzy projektu skupili się na opracowaniu mechanizmów ochrony przeciwko intruzowi, który próbuje wykorzystać znaleziony w programie błąd.

Jak może być wykorzystany błąd w programie?

Intruz może osiągnąć swój cel na trzy sposoby:

- Wprowadzić i wykonać swój kod (1).
- Wykonać istniejący kod w innej kolejności niż go opracowano (2).
- Wykonać istniejący kod w prawidłowej kolejności, ale ze zmienionymi danymi (3).

Przykładowo, wiele popularnych błędów przepełnienia bufora należy do klasy (1), klasa druga to między innymi tak zwany atak w rodzaju return-to-libc (często zapisywane return2libc - zmiana adresu powrotu tak żeby wskazywał na jakąś funkcję - na przykład biblioteki standardowej - i ew. wstawienie na stos odpowiednich parametrów), błędy klasy (3) występują i są wykorzystywane rzadko, przykładem może być zmiana wartości zmiennej określającej pomyślne przejście autoryzacji.

W odróżnieniu do innych poprawek, PaX nie zabezpiecza przed określonymi atakami z zewnątrz, ale został zaprojektowany jako ochrona przed pewną generalną klasą ataków, do których dochodzi z wnętrza systemu.

• Co nam oferuje PAX?

- Wprowadza **randomizację układu przestrzeni adresowej**, czyli w skrócie ASLR, sprawiając, że ładowany dynamicznie kod, pojawia się pod losowym (a więc nie znanym włamywaczowi) adresem.

- implementuje dodatkowe zabezpieczenie w postaci **niewykonywalnych stron pamięci** (NoExec), przez co uniemożliwia wykonywanie fragmentów pamięci takich jak stos i sterta.

- Daje możliwość wyłączenia ograniczeń w przypadku programów, które celowo modyfikują swój kod (chpax, paxctl)

• Randomizacja przestrzeni adresowej(ASLR)

Jedną z cech poprawki PaX i innych poprawek ochrony pamięci jest randomizacja układu przestrzeni adresowej, czyli w skrócie ASLR (Address Space Layout Randomization). ASLR powoduje, że **do różnych miejsc w pamięci systemu ładowane są różne części programu.** Układ pamięci jest więc zmieniany przy każdym uruchomieniu programu. Nie jest to mechanizm ochronny – nie zapewnia żadnej metody kontroli. Dzięki niemu możemy jednak znacznie utrudnić wykorzystanie dziur w naszym systemie zabezpieczeń.
Większość ataków opiera się bowiem na możliwości przewidzenia pewnych adresów w atakowanym procesie. Jeżeli te adresy będą losowe, to intruz będzie musiał je zgadywać. Zgadywanie powinno zaś spowodować całą długą serię łatwych do zaobserwowania przez administratora sięgnięć do nieprzydzielonej pamięci. Niestety PAX ani żadna z popularnych łat nie dodaje do jądra mechanizmu wychwytywania i automatycznego reagowania na takie przypadki.
Poprawki tego typu różnią się między sobą głównie pod względem ich stopnia randomizacji. Generalnie, im wyższy stopień randomizacji, tym trudniej wykonać atak na nasz system. Obecnie poprawka PaX zapewnia użytkownikom najlepszy dostępny stopień randomizacji w systemie Linux, a jednocześnie jest dużo lepsza niż w systemie OpenBSD.
Efekty uboczne stosowania ASLR to fragmentacja przestrzeni adresowej i wyczerpanie systemowych zasobów entropii.

Jak to jest implementowane?

Dla każdego zadania inicjowane są losowo trzy zmienne: delta_exec, delta_mmap, delta_stack. W zależności miejsca może być stosowana inna metoda zabezpieczenia.

RANDMMAP – rejony obsługiwane przez wywołanie do_mmap(), głównie programy typu ELF oraz biblioteki.
Ładowanie bibliotek odbywa się przez wywołanie funkcji mmap(), która może, ale nie musi podać potrzebny do alokacji offset. Jeśli go nie poda, system operacyjny wylicza go zawsze w ten sam sposób, poczynając od pewnego predefiniowanego adresu. Dlatego dla każdego uruchomienia programu biblioteki ładowane są w to samo miejsce. Pax powoduje natomiast, że wspomniany predefiniowany adres jest zmieniany za pomocą losowej zmiennej delta_mmap.

Losowość wprowadzana jest również w funkcji load_elf_binary(), która obsługuje ładowanie plików ELF typu ET_DYN. Wówczas ładowanie odbywa się podobnie do plików ET_EXEC, które ładowane są pod stały adres, w tym przypadku do adresu bazowego dodawana jest jednak losowa wartość delta_exec.

RANDKSTACK – dotyczy stosu jądra, przelosowanie wskaźnika stosu każdego zadania odbywa się za każdym razem w chwili powrotu z wywołania systemowego do przestrzeni użytkownika.

RANDUSTACK – dotyczy stosu użytkownika. Tu inaczej niż w przypadku stosu jądra wskaźnik losowany jest jedynie na samym początku i jest wobec tego wrażliwy na ataki wykorzystujące wycieki informacji.

RANDEXEC – obsługuje pliki ELF typu ET_EXEC, które czynią założenia, że będą ładowane pod stałe adresy. Randomizacja tego adresu nie jest łatwa; sytuacja w której komputer próbowałby odnieść się do instrukcji lub danych, które powinny być pod oryginalnym adresem, zgłosiłaby błąd. Aby skorzystać z zalet losowości, stosowane są wymyślne techniki, takie jak np użycie dwóch osobnych kopii mapowań, które się wzajemnie mirrorują. Pierwsze z nich znajduje się na oryginalnych pozycjach, ale jest zagwarantowane, że te miejsca nigdy są uruchamialne. W przypadku odwołania do takiego miejsca, zostanie wygenerowany błąd, którego – odpowiednio na to przygotowana – obsługa wywoła przeniesienie potoku wywołania do losowo już umieszczonego w pamięci obszaru, gdzie znajduje się kopia pożądanego kodu.

• Niewykonywalne strony pamięci.

Wprowadzenie kodu do istniejącego procesu można wykonać na dwa sposoby: albo zamapować (z prawem wykonywania) do przestrzeni adresowej procesu nową stronę albo zmodyfikować którąś z istniejących stron kodu procesu.
Pierwszy sposób jest trudny do praktycznej realizacji, natomiast próbę skorzystania z drugiego sposobu można łatwo utrudnić lub nawet uniemożliwić przez zablokowanie możliwości przydzielania odpowiednich uprawnień. PAX powoduje, że dane wprowadzane do przestrzeni adresowej procesu będą niewykonywalne, niewykonywalne będą stos i sterta, zablokowane będzie mapowanie stron jednocześnie z prawami pisania i wykonywania oraz modyfikowanie takich stron, a także przechodzenie ze stanu w którym możliwe jest pisanie do stanu wykonywania i odwrotnie.

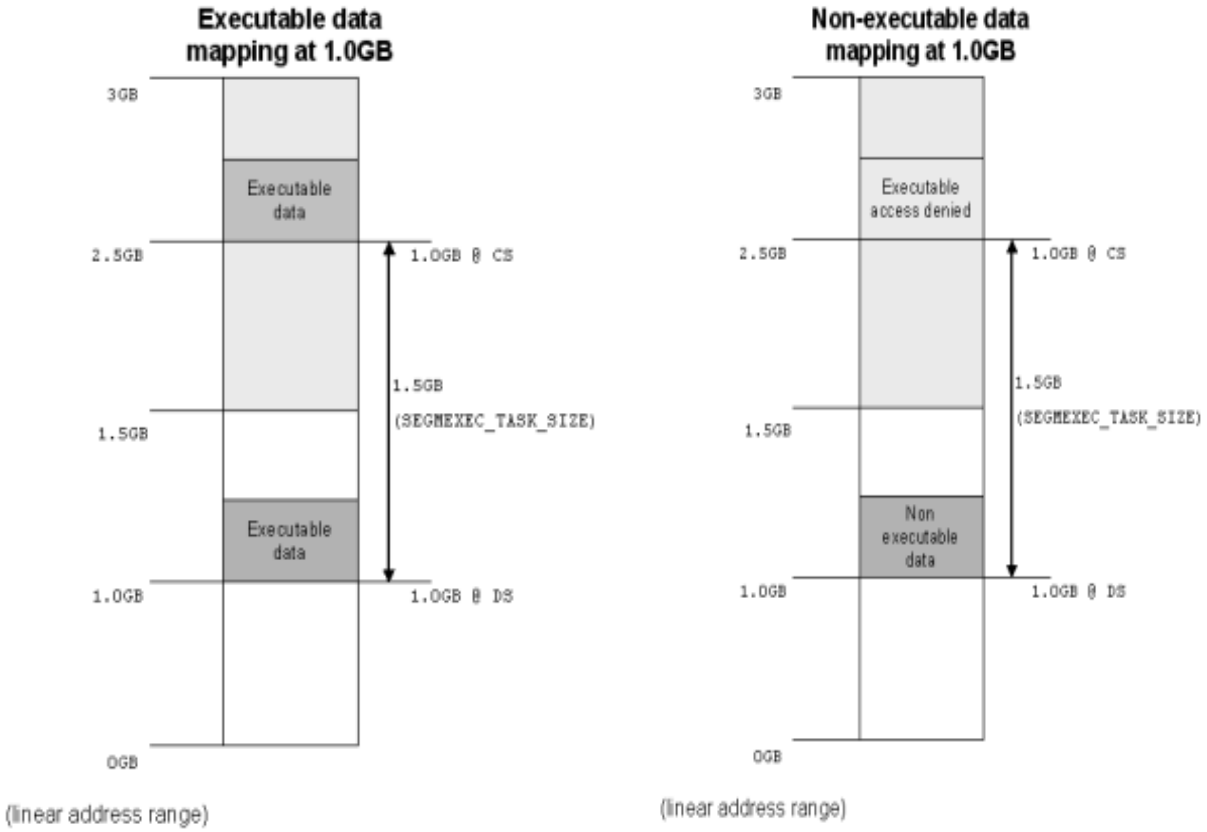
Bit NX (NoExecute)

Implementacja tego zabezpieczenia na większości procesorów nie jest skomplikowana - po prostu w opisie każdej strony w katalogach stron istnieje dodatkowy (prócz bitów praw czytania i pisania) bit pozwalający na określenie praw do wykonywania (zwany często NX od No eXecute). Niestety najbardziej popularne architektury (i386 i powerpc oraz niektóre procesory Intela zgodne z x86-64) tego bitu nie posiadają. Wówczas zabezpieczenie to jest emulowane przez PAX, co jest na pewno mniej wygodne w implementacji, ale równie skuteczne.

Z obsługą bitu NX związane są mechanizmy PAGEEXEC i SEGMEXEC.

SEGMEXEC

Segmexec emuluje działanie bitu NX dzięki wprowadzeniu segmentacji; dzieli się przestrzeń adresową na połowę danych i połowę kodu. Ponieważ wykonywalne fragmenty pamięci mogą być również traktowane jako dane, takie mapowanie jest widoczne w obu segmentach, które wzajemnie się mirrorują. Gdy następuje odwołanie do instrukcji, sprawdzane jest czy dany kod jest mirrorowany, jeśli nie – oznacza to, że dane nie są przeznaczone do wykonywania. Wadą tego rozwiązania jest podział przestrzeni adresowej na połowy, jednak wydajność, w przypadku gdy bit NX musi być emulowany, jest lepsza niż kolejnego.



PAGEEXEC

Jeśli architektura to umożliwia PAGEEXEC wykorzystuje po prostu dostępny bit NX, jeśli nie, jest on emulowany. Do tego celu wykorzystywana jest podręczna pamięć TLB(Translation Lookaside Buffer), którą procesory typu x86 wykorzystują do wyznaczania adresu fizycznego strony. Pamiętane są w niej ostatnio wyznaczone adresy fizyczne stron oraz podstawowe informacje na ich temat, między innymi uprawnienia. Za każdym razem, gdy wyliczany jest adres fizyczny strony, najpierw przeszukiwane jest TLB. Wówczas również PAGEEXEC sprawdza, czy obszar, do którego nastąpiło odwołanie, ma właściwe sobie uprawnienia.

Modyfikacja funkcji mprotect()

Funkcja mprotect() jest odpowiedzialna za zmianę uprawnień obszaru pamięci. Jeśli możliwe jest wykorzystanie lub emulowanie bitu NX, możliwe jest również wprowadzenie ograniczeń dla tej funkcji.
Poprawka PAX ma zagwarantować, że **żadna strona pamięci RAM nie będzie jednocześnie miała praw pisania i wykonywania**. Wyjątkiem są tu mapowania plików typu ELF, które są wykonywane z należytymi prawami dostępu, a więc tylko fragmenty zawierające kod są wykonywalne. PAX zablokuje natomiast mapowanie stron z prawami zapisu i jednocześnie wykonywalnych, przechodzenie ze stanu wykonywalnego do stanu z prawem pisania oraz ze stanu niewykonywalnego do wykonywalnego.

W tym celu użyte będą cztery flagi w polu vm_flags, mianowicie VM_WRITE, VM_MAYWRITE, VM_EXEC oraz VM_MAYEXEC. Ze względu na te flagi konkretne mapowanie może być w 16 stanach, przy czym dozwolone jest tylko 6 z nich, z których z kolei jedynie na 4 zezwalają mechanizmy samego jądra.

Ostatecznie "**legalne**" są następujące **stany**:

VM_MAYWRITE
VM_WRITE | VM_MAYWRITE
VM_MAYEXEC
VM_EXEC | VM_MAYEXEC

Zatem **rozdzielone będą prawa pisania i wykonywania**.

- Jądro tworzy trzy **rodzaje mapowań:**

anonimowe - np. stos, sterta
- tworzone są z flagami VM_WRITE | VM_EXEC | VM_MAYWRITE | VM_MAYEXEC, co nie jest dozwolone przez Pax.
- mprotect() zmienia flagi na VM_WRITE | VM_MAYWRITE

pamięci dzielonej - nie jest konieczna interwencja, stan VM_WRITE | VM_MAYWRITE jest poprawny.

mapowania plików - zależnie od tego, czy w wywołaniu mmap() zażądano prawa pisania, czy nie, ustawiane są flagi VM_WRITE | VM_MAYWRITE bądź VM_EXEC | VM_MAYEXEC; zatem niemożliwe jest jednoczesne ustawienie praw pisania i wykonywania.

Można się zastanawiać, czy takie restrykcje są uzasadnione. Wyobraźmy sobie sytuację, gdy dana strona z zapisanym kodem uzyskuje prawa wykonywania. Możliwy byłby wówczas następujący scenariusz ataku: zapisanie obcego kodu, ret2libc::ret2mprotect(), wykonanie kodu.

Co gdy musimy modyfikować kod w trakcie wykonania ?

Aby korzystać z naszych programów na jądrze z poprawką PaX, nie musimy ich kompilować ponownie. Większość programów będzie działać prawidłowo bez żadnych modyfikacji. Problemy stwarzają biblioteki oraz programy, które modyfikują swój kod w trakcie

wykonania(na przykład standardowa maszyna wirtualna Javy oraz niektóre inne interpretery) Aby zdefiniować wyjątki dla programów tego typu, można użyć narzędzi **paxctl** lub **chpax**. Wówczas jego ustawienia zachowywane są w pliku wykonywalnym. Po uruchomieniu pliku PaX wykrywa te ustawienia i wyłącza niektóre z testów kontrolnych.

• Co jeszcze oferuje nam Pax – PaXTest

Paxtest to zestaw programów, z których każdy sprawdza jeden z funkcjonalnych aspektów PaX. Jeden z nich zapisuje kod maszynowy w ciągu znakowym, a następnie próbuje wykonać ten kod. Prawidłowo działająca konfiguracja z Pax powinna wykryć problem i zatrzymać wykonywanie programu. Inne programy sprawdzają kolejno wszystkie inne funkcje ochronne poprawki PaX. W pakiecie znajdują się także testy sprawdzające randomizację ASLR. Dzięki temu otrzymujemy w miarę pełny obraz poziomu zabezpieczeń. Im więcej testów spowoduje wstrzymanie działania programu, tym lepiej.
Dodatkową zaletą pakietu Paxtest jest możliwość przetestowania także innych poprawek zapewniających ochronę pamięci. Paxtest można pobrać ze strony internetowej znajdującej się pod adresem http://www.adamantix.org/paxtest/, uruchomić i porównać wyniki. Po uruchomieniu Paxtestu na systemie z łatką execshield otrzymamy wyniki, pokazujące, że ochrona przy zastosowaniu execshield jest niewystarczająca.

• Projekty, których częścią składową jest PaX:

Adamantix – znany wcześniej jako Trusted Debian v.1.0. Dystrybucja, która ma być wysoce stabilną, a jednocześnie używalną odmianą Linuksa.

Hardened Gentoo – wydzielona spośród programistów Gentoo grupa, zajmująca się przede wszystkim zagadnieniami bezpieczeństwa i stabilności dystrybucji. Poza PaX'em w Hardened Gentoo korzysta się z osiągnięć SELinuksa, RSBAC oraz innych podprojektów związanych z bezpieczeństwem.

GrSecurity – podstawowy dla bezpieczeństwa zestaw łatek
Elementy i koncepcje pochodzące z PaX stosowane są także w systemie OpenBSD.

• Inne mechanizmy ochrony łączone z PaX

• SSP (Stack Smashing Protector - Przełącznik ochrony stosu)

Przełącznik ochrony stosu (SSP) to poprawka dla kompilatora gcc zabezpieczająca system przed grupą ataków zwanych pod nazwą błędów przepełnienia stosu.

SSP w chwili wykrycia potencjalnie niebezpiecznej funkcji dołącza "minę-pułapkę" do stosu (ponieważ niestety mechanizm detekcji nie jest jeszcze niezawodny), zmienia kolejność lokalnych zmiennych w taki sposób, aby podejrzane zmienne znalazły się obok "miny-pułapki", zwiększając w ten sposób prawdopodobieństwo wykrycia błędu. "Mina-pułapka" często w żargonie określana jest mianem kanarka (w kopalniach kanarki wykrywały kiedyś

obecność śmiercionośnego tlenku węgla, który zabijał kanarki zanim zabił górników). Kanarkiem jest w naszym przypadku losowa liczba umieszczana na stosie. Atak przepełnienia stosu tę liczbę, która jest następnie wykrywana przez SSP przed wykonaniem kodu wykorzystującego ten błąd. Jeżeli SSP wykryje zastąpienie losowej liczb inną liczbą, dokonuje zapisu komunikatu w dzienniku systemu i przerywa pracę programu.

• ACL (Access Control List - Listy Kontroli Dostępu)

ACL czyli listy kontroli dostępu to ułatwienie dla administratora, pozwalające na kontrolowanie uprawnień. Tradycyjne prawa dostępu (odczyt, zapis, wykonanie) pokazują podstawę mechanizmu sterowania dostępem do różnych plików w oparciu o dziewięć bitów plus dwa bity na każdy plik. Zaletą tego rozwiązania jest absolutna prostota.
Niestety, jeżeli administrator systemu będzie chciał ustawić wszystkie możliwe uprawnienia dostępu do pliku n użytkownikom, będzie potrzebował 2n różnych grup. Jest to oczywiście niepraktyczne, zatem nowsze systemy operacyjne wykorzystują pojęcie grupy rozszerzonej - listy kontroli dostępu – ACL, która umożliwia bardzo szczegółową kontrolę uprawnień w porównaniu z tradycyjnym modelem uprawnień.

Przy użyciu list kontroli dostępu system operacyjny dołącza do pliku dodatkową listę uprawnień dla określonych użytkowników i grup. Umożliwia to przypisanie uprawnień odczytu lub zapisu dla pojedynczego pliku dwóm lub trzem użytkownikom, a nie tylko całej grupie.

• RBAC (Role Based Access Control)

W modelu zwanym TE (ang. Type Enforcement) każdy proces w systemie otrzymuje atrybut zwany domeną, a każdy obiekt - otrzymuje atrybut zwany typem. Wszystkie procesy będące w tej samej domenie są traktowane jednakowo (mają jednakowe uprawnienia), jak również wszystkie obiekty o danym typie są traktowane na równi. Dla każdej pary domeny i typu możemy określić, jakie działania mogą być podejmowane. Domena może być powiązana z kilkoma punktami wejścia - programami, które mogą być wykonywane przez użytkowników spoza domeny, a także wybranymi programami pomocniczymi i/lub bibliotekami, które są używane w ramach domeny. Pozwala to np. oddzielić programy zajmujące się pocztą od innych części systemu.

Tradycyjny model RBAC (ang. Role-Based Access Control) spotykany w systemach UNIX pozwala użytkownikom spełniać określoną role i dostarcza jednego zestawu uprawnień dla tej roli. Jest to sytuacja odmienna do tej, spotykanej w systemie SELinux, gdzie użytkownik może posiadać nie jedną rolę, ale zestaw ról, a uprawnienia każdej z nich mogą być definiowane zestawem domen.

Bibliografia

http://pax.grsecurity.net/docs/index.html
http://en.wikipedia.org/wiki/PaX
www.linux-magazine.pl/issue/02/CoverStory_PAX.pdf