UNIX: architektura i implementacja mechanizmów bezpieczeństwa

Wojciech A. Koszek
dunstan@FreeBSD.czest.pl
Krajowy Fundusz na Rzecz Dzieci

Plan prezentacji:

- Wprowadzenie do struktury systemów rodziny UNIX
- Prezentacja funkcjonalności wraz z mechanizmami kontroli uprawnień
- System z punktu widzenia osoby zainteresowanej atakiem na system (bądź obroną poprzez audyt)
- Podsumowanie: możliwe drogi zapobiegania ewentualnym skutkom wykorzystania błędów w oprogramowaniu

UNIX: Aplikacje

- Jeden z możliwy podziałów aplikacji:
  - skrypty powłoki: *#!/path/to/shell*
  - pliki binarne (prawie zawsze w formacie ELF):
    - nagłówki z informacjami dotyczącymi architektury, odpowiednich struktur (informacje o stosie, stercie), informacje do debuggingu, informacje dotyczące bibliotek funkcji
    - właściwa zawartość w postaci instrukcji procesora
- Po uruchomieniu widziane jako procesy

Aplikacje: kontrola uprawnień

- Dostęp realizowany z wykorzystaniem testów bazujących na standardowym mechaniźmie uprawnień:
  - UID
  - GID
  - System plików: atrybuty plików (odczyt, zapis, wykonywanie)

UNIX: Przestrzeń jadra

- Dostęp do wszystkich zasobów komputera
- Posiada “globalny” obraz pamięci maszyny: to tutaj odbywa się właściwa alokacja zasobów, manipulacja strukturami procesów, wątków, scheduling, mapowanie deskryptorów plików na właściwe obiekty
- Musi udźwignąć ciężar całej funkcjonalności systemu udostępniając najbardziej podstawowe mechanizmy

Przestrzeń jądra: kontrola uprawnień

- Podłożem do kotroli uprawnień jest sprzętowa architektura sprzętu:
  - Tryby pracy procesora
- W warstwach wyższych interpretacja znanych uprawnień:
  - UID
  - GID
  - Inne

Przestrzeń użytkownika

- Wykorzystuje funkcjonalność jądra tworząc właściwe środowisko użytkowe:
  - komunikacja poprzez wywołania systemowe (*ang. syscall*)
  - większość wywołań są celem samym w sobie: read()/write() itd.
  - Inne są uniwersalnym sposobem na interakcję z systemem np.: ioctl(2), sysctl(3)

Kontrola wywołań systemowych:

- Niektóre niedostępne dla zwykłego użytkownika:
  - ładowanie modułów jądra, ustalanie nazwy maszyny
- Inne dostępny w przypadku poprawnych argumentów:
  - kill(2)
- Inne dostępne w zależności od kontekstu:
  - ioctl(2)

Wywołania “uniwersalne”: ioctl(2)

- Poprzez operację na dostarczonym deskryptorze możliwe jest:
  - Przekazanie do jądra unikalnego identyfikatora żądania oraz argumentów w postaci bufora danych
  - Odebranie z przekazanego bufora danych pochodzących z przestrzeni jądra
- Niemożliwe jest bezpośrednie sięganie do danych wskazywanych przez wskaźniki w buforze:
  - Potrzebne są operacje kopiowania między jądrem i użytkownikiem: copyin(9)/copyout(9)

Cechy wspólne wspomnianych mechanizmów:

- Ograniczają się do testów na identyfikatorze użytkownika wykonującego daną akcję
- Wszystkie zawierały w swojej historii błędy umożliwiające dokonanie ataku na aplikację i system: interpretacja binariów, brak kontroli argumentów przekazywanych z przestrzeni jądra do użytkownika, problemy z operacjami dotyczącymi deskryptorów, sygnałów, procesów

Przykłady niebezpiecznych zależności w obrębie systemu:

- Funkcja<->funkcja:
  - Znane przez wszystkich buffer overflow, format strings
- Aplikacja<->aplikacja:
  - Wymagany jednolity format komunikacji (np.: dane przesyłane w postaci protokołu)
- Aplikacja<->system:
  - Zmienne środowiskowe, wartości parametrów globalnych dla całego systemu

Przykłady błędów mających wpływ na bezpieczeństwo:

- *ifconf()*
- *if_clone_list()*
  - Brak kontroli wartości zmiennej przekazanego w strukturze będącej argumentem. Przekazanie wartości ujemnej to DoS (Denial of Service)
- PECOFF:
  - Spreparowanie pliku w odpowiedni sposób powoduje w przestrzeni jądra uzyskanie ujemnej wartości zmiennej. Rezultat: Denial of Service

Główne powody problemów:

- Złożoność wewnętrznych struktur danych jest ogromna, a ich analiza trudna
- Rosnące wymagania stawiene systemowi UNIX (wsparcie dla SMP, przenośność, kompatybilność z innymi systemami) spowodowały, iż kontrola bazująca na UID/GID jest niewystarczająca
- Błędy nieuważnych programistów: nie wolno ufać danym przekazywanym do systemu bądź aplikacji!

Jak strzec się przed problemami z bezpieczeństwem?

- Poprzez tworzenie bezpiecznego oprogramowania
- Przez audyt bezpieczeństwa oprogramowania
- Poprzez wykorzystanie zaawansowanych możliwości nowoczesnych systemów operacyjnych
  - zastosowania metod minimalizujących straty w przypadku pomyślnego ataku

Audyt: cechy wspólne ataków

- Wykorzystanie niepoprawnej (braku) kontroli typów i wartości zmiennych
  - Przekazanie wartości nieprzewidzianych przez programistę danego oprogramowania
- Spowodowanie niespotykanego przepływu instrukcji w oprogramowaniu:
  - Interaktywne interfejsy często mogą zostać wykorzystanie nieinteraktywnie

Metodologie audytowania

- Typowe podejście do problemu audytu bezpieczeństwa oprogramowania:
  - Analiza kodu binarnego aplikacji
  - Analiza kodów źródłowych aplikacji
- Podejście mniej popularne, aczkolwiek skuteczne: analiza typu brute-force

Zaawansowane mechanizmy bezpieczeństwa:

- Dodatkowa kontrola w warstwie jądra systemu
- Możliwość tworzenia polityk koniecznych do wykonania ściśle określonego zadania
- Możliwość limitowania zasobów dostępnych dla aplikacji

Minimalizacja uprawnień

- Capabilities/Privileges:
  - Zminimalizowanie uprawnień koniecznych do wykonania uprzywilejowanej akcji:
  - Wymagana funkcjonalność nie wymusza na administratorze uruchamiania aplikacji z UID == 0
- Access Control Lists
  - Szczegółowe określenie praw dostępu do plików

Kryteria kontroli

- Testy w oparciu o GID/UID przestały wystarczać
- Systemy muszą posiadać dodatkowe informacje poddawane kontroli:
  - Rozszerzone atrybuty w warstwie procesów i systemie plików

Przykłady dostępnych rozwiązań:

- FreeBSD
  - TrustedBSD
- GNU/Linux
  - SELinux
- Solaris (OpenSolaris)

Mechanizmy kotroli mają pewne wady:

- Konfiguracja bywa pracochłonna:
  - Łatwość i wygoda administracji systemem maleje wraz ze wzrostem bezpieczeństwa
- Wdrożenie do powszechnego użycia nawet w obrębie jednej maszyny nie zawsze jest łatwe
- Nie wszystko da się kontrolować:
  - system to jedna całość

Podejście alternatywne: separacja i wirtualizacja

- Zamiast modyfikować działanie aplikacji, możliwe staje się zmodyfikowanie środowiska, w którym aplikacja pracuje:
  - chroot(2) (UNIX)
  - jail(2) (FreeBSD)
  - Zones (Solaris)
- Wirtualizacja systemu operacyjnego:
  - Xen

Dziękuję za uwagę

Wojciech A. Koszek
dunstan@FreeBSD.czest.pl