

Bezpieczeństwo systemów komputerowych

Zaawansowane techniki wywoływania złośliwego kodu

Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

5 grudnia 2011

Wyrzysanie przepelnienia bufora

- ▶ Umieszczenie w przepelnianym buforze kodu wywołujacego powloke systemu operacyjnego (ang. *shellcode*) i podmienienie adresu powrotu z wykorzystaniem slizgania sie po NOP-ach
- ▶ Bezposrednie podmienienie adresu powrotu na stosie i wywolanie funkcji bibliotecznej (lub uslugi systemowej)

Zapobieganie

- ▶ ASLR (ang. *address space layout randomization*):
 - ▶ adresy segmentów kodu, danych i stosu są przydzielane pseudolosowo;
 - ▶ bardzo zmniejsza prawdopodobieństwo skutecznego wykonania shellcode'u lub wywołania funkcji bibliotecznej
- ▶ $W \oplus X$
 - ▶ strona pamięci nie może mieć jednocześnie atrybutów zapisywalności (ang. *Writable*) i wykonywalności (ang. *eXecutable*);
 - ▶ nazwa jest błędna, powinna być $\neg(W \wedge X)$;
 - ▶ uniemożliwia wykonanie shellcode'u.

Praktyka stosowania ASLR

- ▶ Większość dystrybucji Linuksa:
 - ▶ domyślnie randomizuje tylko adresy segmentów stosu, sterty i kodu bibliotek dzielonych;
 - ▶ umożliwia opcjonalnie włączenie randomizowania adresów segmentów kodu aplikacji;
 - ▶ randomizowanie adresów w kodzie aplikacji wymaga skompilowanie z ustawioną opcją `-pie` (ang. *position independent executable*);
 - ▶ opcja `-pie` zmniejsza wydajność kodu.
- ▶ Windows Vista, 7:
 - ▶ domyślnie randomizuje tylko adresy segmentów stosu i sterty;
 - ▶ umożliwia opcjonalnie włączenie randomizowania adresów segmentów kodu bibliotek i aplikacji;
 - ▶ aplikacja i wszystkie biblioteki, z których korzysta muszą być „ASLR compatible”;
 - ▶ jest wiele aplikacji niekompatybilnych.
- ▶ Mac OS 10.6:
 - ▶ randomizuje tylko adresy segmentów kodu bibliotek dzielonych.

Ograniczenia stosowania ASLR

- ▶ Zbyt mała entropia źródła pseudolosowości umożliwia wykonanie ataku z prawdopodobieństwem istotnie większym od zera.
- ▶ Funkcja `strcpy` zwraca w rejestrze `rax` adres bufora, który przepełniła.
- ▶ W architekturze x86 jest instrukcja `jmp rax`.
- ▶ Jeśli w jakiejś aplikacji wystąpi koincydencja tych dwóch zdarzeń, randomizacja nie zapobiegnie wywołaniu kodu umieszczonego w buforze.
- ▶ Wiele podobnych sztuczek daje się wymyślić.

Praktyka stosowania $W \oplus X$

- ▶ W architekturze x86 realizowane za pomocą atrybutu NX (ang. *no execute*) strony pamięci.
- ▶ W wielu dystrybucjach Linuksa używanie atrybutu NX jest domyślnie wyłączone.
- ▶ Jeśli sprzęt nie wspiera NX, dla Linuksa istnieją łątki emulujące $W \oplus X$.
- ▶ Windows 7 domyślnie włącza $W \oplus X$ dla plików wykonywalnych i bibliotek oznaczonych jako „ $W \oplus X$ compatible”.
- ▶ Jest wiele aplikacji niekompatybilnych.

Ograniczenia stosowania $W \oplus X$

- ▶ Chroni tylko przed wykonaniem kodu wstrzykniętego w czasie wykonania.
- ▶ Nie chroni przed wykonaniem kodu już załadowanego do pamięci.

ROP

- ▶ Return Oriented Programming
- ▶ Gadżet to fragment kodu zakończony instrukcją `ret`, końcowy fragment funkcji lub procedury.
- ▶ Gadżety można wywoływać, umieszczając na stosie ich adresy.
- ▶ Instrukcja `ret` na końcu gadżetu sprawia, że wywołany zostaje kolejny gadżet.
- ▶ Problem: znaleźć w kodzie wykonywalnym zbiór gadżetów, który umożliwi zaimplementowanie dowolnego algorytmu.

Przykład

- ▶ Gadżety

```
addr_g1: pop rax  
         ret
```

```
addr_g2: pop rbp  
         ret
```

```
addr_g3: mov [rbp], rax  
         ret
```

- ▶ Nadpisujemy fragment stosu

```
addr_g3
```

```
addr0
```

```
addr_g2
```

```
value0
```

```
addr_g1 ; Tu był oryginalny adres powrotu.
```

- ▶ Po wykonaniu w pamięci pod adresem `addr0` znajdzie się wartość `value0`.