

Übung 10

Assembler

. . .

;function2:

8049775: 55 push ebp

8049776: 89 e5 mov ebp,esp

8049778: 53 push ebx

8049779: 83 ec 04 sub esp,0x4

804977c: e8 5f 00 00 00 call 80497e0 <__x86.get_pc_thunk.ax>

8049781: 05 7f 18 09 00 add eax,0x9187f

8049786: 83 ec 0c sub esp,0xc

8049789: 8d 90 08 20 fd ff lea edx, [eax-0x2dff8]

804978f: 52 push edx

8049790: 89 c3 mov ebx,eax

8049792: e8 59 79 00 00 call 80510f0 <_IO_puts>

8049797: 83 c4 10 add esp,0x10

804979a: 90 nop

804979b: 8b 5d fc mov ebx, DWORD PTR [ebp-0x4]

804979e: c9 leave

804979f: c3 ret

;function1:

80497a0: 55 push ebp

80497a1: 89 e5 mov ebp,esp



80497a3: 53 push ebx

80497a4: 83 ec 14 sub esp,0x14

80497a7: e8 34 00 00 00 call 80497e0 <__x86.get_pc_thunk.ax>

80497ac: 05 54 18 09 00 add eax,0x91854

80497b1: 83 ec 0c sub esp,0xc

80497b4: 8d 55 ee lea edx,[ebp-0x12]

80497b7: 52 push edx

80497b8: 89 c3 mov ebx,eax

80497ba: e8 a1 77 00 00 call 8050f60 <_IO_gets>

80497bf: 83 c4 10 add esp,0x10

80497c2: 90 nop

80497c3: 8b 5d fc mov ebx, DWORD PTR [ebp-0x4]

80497c6: c9 leave

80497c7: c3 ret

;main:

80497c8: 55 push ebp

80497c9: 89 e5 mov ebp,esp

80497cb: 83 e4 f0 and esp,0xfffffff0

80497ce: e8 0d 00 00 00 call 80497e0 <__x86.get_pc_thunk.ax>

80497d3: 05 2d 18 09 00 add eax,0x9182d

80497d8: e8 c3 ff ff ff call 80497a0 <function1>

80497dd: 90 nop

80497de: c9 leave

80497df: c3 ret



```
C
 1 #include <stdio.h>
 2 void function2()
 3 {
 4
       printf("Hello World\n");
 5 }
 6
 7 void function1()
8 {
 9
      char buffer[10];
      gets(buffer);
10
11 }
12
14 void main()
15 {
16
      function1();
17 }
```

Gegeben ist ein C-Programm und der daraus resultierende x86 32bit Assembler-Code

```
(Konsolenbefehle: gcc -m32 -static uebung12.c -o uebung12 objdump -M intel -D uebung12) (Tipp: https://c9x.me/x86/)
```

- 1. Was macht das Programm?
- 2. Erläutern Sie warum aus der Anweisung



call 0x80497a0

der Opcode

e8 c3 ff ff ff

resultiert

- 3. Zeichnen Sie den Stackframe für "function1". Gehen Sie dabei davon aus, dass die Rücksprungadresse der Funktion "main" an der Speicherstelle Oxfffffff4 liegt und die Programmausführung an Adresse 0x80497b4 gestoppt wurde.
- 4. Das Beispielprogramm enthält eine sog. "Bufferoverflow" Sicherheitslücke. Erarbeiten Sie eine Zeichenkette, deren Übergabe an das Programm die Ausführung derartig beeinflusst, dass "function2" aufgerufen wird.

Notieren

Sie das Beispiel in hexadezimaler Darstellung. Ein anschließender Absturz

des Programmes kann dabei in Kauf genommen werden!

5. Welche Schlüsse ziehen Sie aus diesem Aufgabenblatt in Bezug auf die Verwendung der Funktion "gets"? Bzw. wie können Sie den Code sicherer gestalten?