Rechnersicherheit Übung 1

Alexander Steen , Max Wisniewski

Tutorium : Do 10 - 12

In unserem Programm haben wir zunächst die einfach Funktion f geschrieben.

```
1 int f(int a, int b){
2   return a+b;
3 }
```

Dies erfüllt die Anforderungen an eine *einfache* Funktion und sie benutzt keine relativ addressierten Funktionen (wie printf), kann also nachher vom Heap ausgeführt werden. Damit wir die Funktion leichter in den Heap laden können, legen wir direkt unter unsere Funktion noch eine leere *Dummyfunktion*. Nun können wir uns die Differenz der beiden Funktoisnpointer betrachten und wissen daher, wie lang die Funktion f ist.

Da die Funkionen am Anfang richtig herum in den Stack geladen werden, müssen wir von der Funktion g (also der Addresse der Funktion) den der Funktion f abziehen, da g weiter unten eine höhere Nummer besitzt.

```
1  int func_size;
2  int (*func_pointer)(int, int);
3  char *function_dump;
4
5  func_size = &g - &f;
6  function_dump = (void *)malloc(func_size);
7  memcpy(function_dump, &f, func_size);
8
9  int i;
10 printf ("Aus dem Heap:\n");
11 for (i = 0; i < func_size; ++i){
12  printf ("Adresse %p \t %#x \n",function_dump+i,
13                          * ((unsigned char *)(function_dump+i)));
14 }
```

Haben wir die Länge, können wir uns mit *malloc* Speicher auf dem Heap reservieren und mit *memcpy* kopieren wir die Binärdaten der Funktion auf den Heap. Die größe stimmt mit der vorher ermittelten größe überein, da alle Befehle genau ein Byte einnehmen.

Nun können wir über unseren Speicher iterieren. Um die Darstellung korrekt in Hex auszugeben, müssen wir die Einträge auf *unsignded char* casten, da die Darstellung sonst mehrer F's vor die eigentlichen Werte schreibt. Wollen wir nun die Funktion ausführen, laden wir auf einen angelegten Funktionspointer die Addresse Startaddresse des allokierten Speichers.

Damit das Aufgeht, müssen wir den Speicherpointer ersteinmal auf einen Funktionspointer casten.

Haben wir das getan, können wir den Pointer wie bei einem normalen Funktionsaufruf benutzen.

```
1 func_pointer = (int (*)(int, int))function_dump;
2 int ergebnis = func_pointer(2,4);
3 printf("Ergebnis: %d\n",ergebnis);
```

Dies führt nun, wenn man es normal kompiliert, an dieser Stelle zu einem *Segmentation Fault*, da standardmäßig für den Heap ein *No-Execution-Bit* gesetzt ist.

Dies kann man leicht umgehen, indem man beim Kompilieren *gcc* den Parameter *-Wa,–execstack* mitgibt.

Einmal ausführen zeigte uns, dass der Hexcode demjenigen entspricht, den einem auch gdb ausgibt, wenn man den Code disassembled und in Hex ausgibt.

Das Ergebnis ist auch, wie erwartet 6. Das selbe hat uns auch ein normaler Aufruf von f am Anfang der Entwicklung geliefert.