Betriebssysteme, Übung 5

Prof. Dr. Jan Dünnweber

Verteilte Systeme und Betriebssysteme

Funktionszeiger

- Das Return-Statement hat für Intel-Prozessoren den OpCode `0xc3`
  - Schreiben Sie eine Funktion, die einen Funktionszeiger als Parameter bekommt und durch Suche nach dem Return-Statement (z. B. mit `memcmp`) feststellt, wieviele Byte die Funktion hat
  - Schreiben Sie eine Funktion, die einen Funktionszeiger als Parameter bekommt und den Maschinencode dieser Funktion in einer Datei speichert
- Schreiben Sie ein C Programm, das eine Funktion, deren Maschinencode in einer Datei gespeichert ist, lädt und aufruft
  - Der Speicherbereich mit dem Code muss mit `mmap()` auf einer Adresse reserviert sein, die ein Vielfaches der Seitengrösse Ihres Systems ist (mit `sysconf(_SC_PAGE_SIZE)` bestimmen)
  - Um reservierten Speicher als ausführbar zu kennzeichen, müssen Sie `mprotect()` aufrufen ($\rightarrow$ *man*-page)

Dieses Programm enthählt keine Fehler

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

int main(int argc, char **argv)  {
  char *p = malloc(1);
  *p = 'a';
  char c = *p;
  printf("\n␣␣[%␣c␣]\n" , c);
  free(p);
  return 0;  }
```

- Überprüfen Sie den Code mit `valgrind` *Memcheck*

Freigabe einer nicht reservierten Adresse

Der Code wird nun modifiziert

```
#include <stdlib.h>
#include <string.h>
int main(int argc, char **argv) {
  char *cp;
  cp = (char *)malloc(50);
  cp = "ohohoh";
  free(cp);  return 0; }
```

- Sehen Sie sich erneut die Ausgabe von `valgrind` *Memcheck* an

Zugriff auf eine bereits freigegebene Adresse

Auch dieses Code-Beispiel ist fehlerhaft

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

int main(int argc, char **argv) {
  char *p = malloc(1);
  *p = 'a';
  char c = *p;
  printf("\n␣[%c]\n", c);
  free(p);
  c = *p;
  return 0;  }
```

- Sehen Sie sich die Ausgabe von `valgrind` *Memcheck* zu dem neuen Fehler an

Zugriff außerhalb des reservierten Speichers

Es wird nur 1 Byte reserviert

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

int main(int argc, char **argv) {
  char *p = malloc(1);  *p = 'a';
  char c = *(p + 1);
  printf("\n␣[%c]\n", c);
  free(p);
  return 0;  }
```

- Sehen Sie sich auch hierzu die Ausgabe von `valgrind` *Memcheck* an, korrigieren Sie *alle* Fehler (auf Folie 4–6) und überprüfen Sie die Ausgabe, wenn Sie das `free` weglassen