Typische Speicherfehler in C

Thorsten Ploß

Informatik Universität Hamburg
Proseminar: C-Grundlagen und Konzepte

17.05.2013

Inhaltsverzeichnis

Einleitung

Speicherfehler in C sind ein sehr relevantes Thema:

- Sehr direkter Einfluss auf den Speicher
  - Fehler bieten Angreifern Schwachstellen
  - Zufällig können Daten verändert werden
- Wenige Automatismen zur Speicherverwaltung
- Einsatz in heiklen Situationen
  - Lebensgefahr
  - Große Geldverluste
- Fehlerquellen oft nur sehr aufwendig ermittelbar

Nicht initialisierte Variablen

- Kein definierter Wert nach der Deklaration
- Kein Zwang zu initialisieren
- Je nach Compiler unbemerkt
- bei Pointern besteht die Gefahr, unbestimmte Speicherbereiche zu verändern

```
3  int t;
4  int array[2];
5  char c;//Ein paar Variablen
6
7  int main(int argc, char *argv[]){
8          int i;
9          char b;//Ein paar weitere Variablen
10         printf("%d %d %d %c %d %c", t,array[0],array[1],c,i,b);
11         return(0); //Was da wohl raus kommt?
12 }
```

Der Gültigkeitsbereich

- Variablen sind nur innerhalb ihres Anweisungsblocks erreichbar
  - Folge: Zugriff auf Variable eines anderen Blocks schlägt fehl
- In inneren des Anweisungsblocks Variablen verschatten
  - Neue Variable mit gleichem Namen
  - Verhindert innerhalb des Blocks Zugriff auf die andere Variable
  - Nach dem Block ist die alte Variable unverändert
  - Folge: Werte gehen verloren

```
3  int a = 0;
4
5  int main(int argc, char *argv[]){
6      if(argc > 0){
7          double result = argc/4;
8          int a = 9;
9      }
10     printf("Ein Viertel von argc ist %f \n",result);
11     printf("a ist 9? %d \n",a);
12 ...
```

Das Ende des Array

- Arrays basieren auf Pointern
- Keine Zugriffskontrolle
- Daher versehentlicher Zugriff auf andere Speicherbereiche möglich
- Bestehendes Risiko Einfluss auf die Rücksprungadresse zu nehmen

```
3  int array[5];
4  int t;
5
6  int main(int argc, char *argv[]){
7          t = 42;
8          printf("Die Antwort lautet: %d\n",array[5]);
9          //Ein Index zu weit und die Wahrheit ist berechnet...
10         printf("Irgendwo im Speicher steht %d\n", array[123]);
11         return(0);
12 }
```

Inhaltsverzeichnis

Null-Pointer

- Fehlgeschlagene Allokation
- Fehlende Initialisierung
- Resultate
  - Pointer auf unerwünschte Adresse
  - Versuchter Zugriff auf Kernel-Speicher
  - Segmentation Error

Pointerprobleme

- Adresszuweisung statt Wertzuweisung
  - entsteht schnell durch Tippfehler
  - Resultat: Pointer auf unbekannte Adresse
- Allocation mit unzureichend viel Speicher
  - Annahmen über benötigte Speichermenge
  - int braucht meist mehr als 2 Byte

```
3  int BUFF = 80;
4  int *t,f = 8;;
5
6  int main(int argc, char *argv[]){
7      t = (int *)malloc(2);
8      *t = 4; //In t soll 4 gespeichert werden
9      char *array = (char *)malloc(sizeof(char)*BUFF);//kein
           Chararray der Welt ist laenger als 80 Zeichen!
10     t = &f; //oder soll in t doch lieber 8 gespeichert werden?
11     return(0);
12 }
```

Speicherlecks

- Speicher nicht wieder freigeben
- Bleibt lange unbemerkt
- Kann für programmübergreifende Schwierigkeiten sorgen
- Hinterhältigster aller Speicherfehler

```
3  int *a;
4
5  int main(int argc, char *argv[])
6  {
7  while(1)
8  {
9  a = (int *)malloc(sizeof(int));
10 }
11 return(0);
12 }
```

Benutzung nach Freigabe und Doppelte Freigabe

- Einen Pointer nach seiner Freigabe nutzen
  - Führt zu undefiniertem Verhalten
  - Kann den Wert von anderen Pointern verändern
- Einen Pointer doppelt freigeben
  - Führt zum Laufzeitfehler

```
3  int *p;
4  int *z;
5
6  int main(int argc, char *argv[]){
7          p = (int *)malloc(sizeof(int));
8          *p = 1337;
9          free(p);
10         z = (int *)malloc(sizeof(int));
11         *z = 8;
12         *p = 42;
13         printf("Eine Spinne hat %d Beine\n",*z);
14         free(p);
15         return(0);
16 }
```

Zusammenfassung

- Fehler machen ist leicht
- Fehler beheben ist schwer
- Viel kann Schaden nehmen
- Warnungen sind essentiell
- Werkzeuge wie Valgrind können helfen

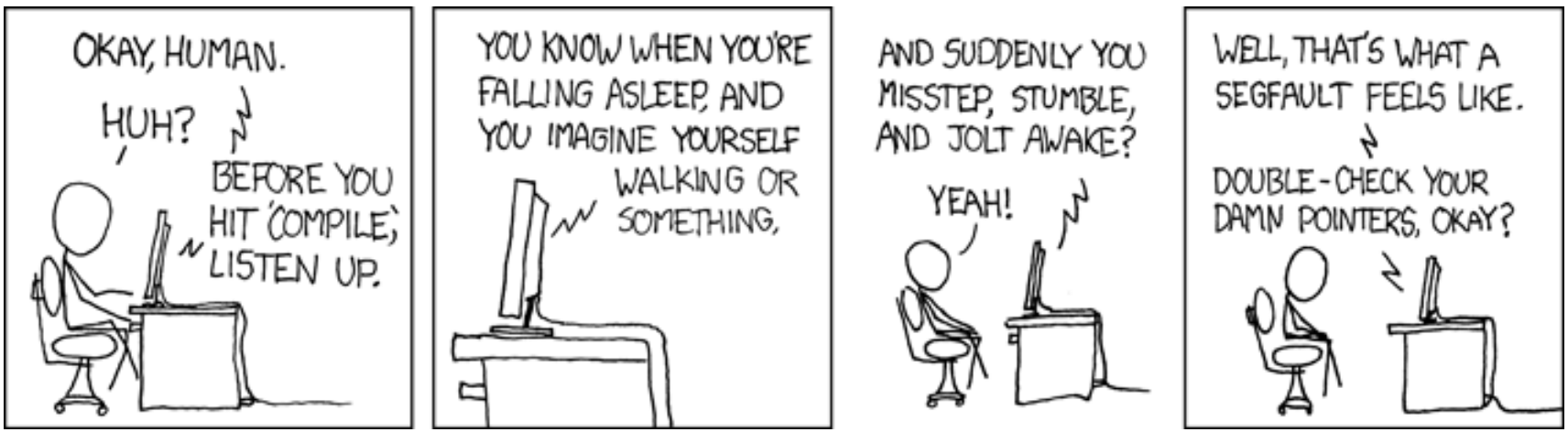

Abbildung: **Compiler Complaint**; Quelle:http://xkcd.com/371/

Quellenverzeichnis

- http://www.peace-software.de/ckurs12.html [Stand: 13.05.2013]
- http://pronix.linuxdelta.de/C/standard_C/c_programmierung_17.shtml [Stand: 13.05.2013]
- http://de.wikibooks.org/wiki/C-Programmierung:_Speicherverwaltung [Stand: 13.05.2013]
- https://de.wikipedia.org/wiki/Speicherleck [Stand: 13.05.2013]
- http://c-buch.sommergut.de/Kapitel10/G%FCltigkeitsbereich-von-Variablen.shtml [Stand: 13.05.2013]
- Bild auf Folie 11: http://xkcd.com/371/ [Stand 13.05.2013]