

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Arbeiten mit Binärdateien und Assembler

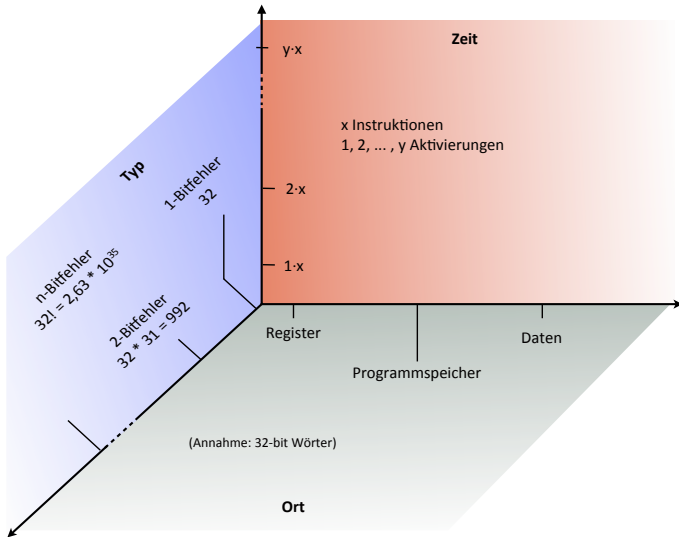
Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

Wintersemester 2020



Fehlerraum



C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
    static int s = 3;
    int x, y;
    char* p =
        malloc(100);
}
```

Assembler-Code

```
4004f0 <main>:
 4004f0: push    %rbp
 4004f1: mov     %rsp,%rbp
 4004f4: sub     $0x10,%rsp
 4004f8: movabs $0x64,%rdi
 400502: callq  4003e0 <malloc@plt>
 400507: mov     %rax,-0x10(%rbp)
 40050b: add     $0x10,%rsp
 40050f: pop     %rbp
 400510: retq
```

Wo können Datenfehler auftreten?

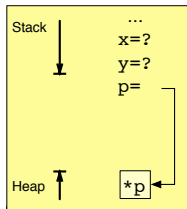
1. RAM: `-0 x10 (% rbp)`
↳ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `% rsp`
3. Sonstige CPU-Register: `% rip , % rflags`



Speicherorganisation und Sektionsgrößen

RAM

Flash / ROM



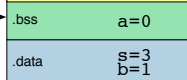
```

int a; // a: global, uninitialized
int b = 1; // b: global, initialized
const int c = 2; // c: global, const

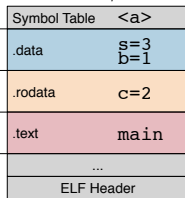
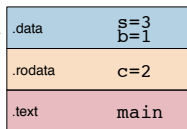
void main() {
    static int s = 3; // s: local, static, initialized
    int x, y; // x: local, auto; y: local, auto
    char* p = malloc( 100 ); // p: local, auto; *p: heap (100 byte)
}
    
```

compile / link

Quellprogramm



copy



flash

µ-Controller

ELF-Binary

```

$ size --format=berkeley a.out
text data bss  dec  hex  filename
2592   8   4 2604 a2c   a.out
    
```

Sektionsgrößen in Byte,
„text“ umfasst hier: .text + .rodata



nm: Ausgabe der Symboltabelle

```
1 00000000000004028 D var_initialized
2 ...
3 00000000000201028 B __bss_start
4 00000000000201028 b var_uninitialized
5 ...
6 0000000000000061a T main
7 00000000000000580 t register_tm_clones
8 00000000000000510 T _start
9 ...
10 0000000000000066d t int max<int,0u>(int,int)
11 0000000000000067c t int max<int,1u>(int,int)
```

■ Nützliche Optionen

- `-C, --demangle`: Dekodieren der C++-Namensmangelung:
`_Z3maxIiLj0EET_S0_S0_` \Rightarrow `int max<int, 0u>(int, int)`
- `-S, --print-size`: Ausgabe der Symbolgrößen
`0000000000000061 a 00000028 T main`

objdump: Ausgabe von Informationen über Objektdateien

```
1  int main(){
2    4007cd:  55                push   %rbp
3    4007ce:  48 89 e5         mov    %rsp,%rbp
4    4007d1:  48 83 ec 10      sub    $0x10,%rsp
5    int a = max<int>(23U, 42);
6    4007d5:  be 2a 00 00 00   mov    $0x2a,%esi
7    4007da:  bf 17 00 00 00   mov    $0x17,%edi
8    4007df:  e8 04 01 00 00   callq 4008e8 <_Z3maxIiET_S0_S0_>
9
10   4007e4:  89 45 fc         mov    %eax,-0x4(%rbp)
11   std::cout << a << "\n";
12   4007e7:  8b 45 fc         mov    -0x4(%rbp),%eax
13   . . .
```

■ Nützliche Optionen

- -S: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
- -D: alle Sektionen disassemblieren



Analyse der Toleranzmaßnahmen

- normalerweise: Fehlerinjektion
- dieses Semester: Fehlerräumenanalyse
 - Einfluss des Speichers
 - Einfluss der Laufzeit

■ Aufrufgraph

`make callgraph_{baseline,tmr,ean}`

■ ELF

`build/{baseline,tmr,ean}.elf`

■ Objektdateien

`build/CMakeFiles/{baseline,tmr,ean}.dir/src/*.o`

