

# Buffer overflow mit Stützrädern

Wie demonstriert man das Prinzip einer buffer overflow Attacke ohne  
Assembler-Kenntnisse vorauszusetzen?

Stefan Huber

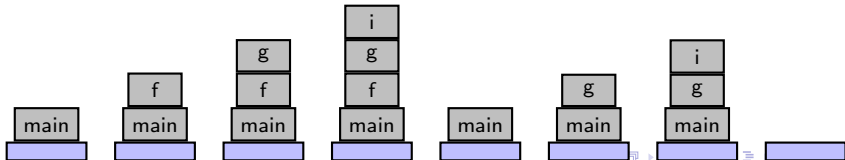
Chaostreff Salzburg

6. Mai 2011

# Basics: call stack

- ▶ Die Hierarchie von Funktionsaufrufen wird im *call stack* abgebildet.
- ▶ Auf dem stack liegen *stack frames*: die lokalen Variablen einer Funktion (und ggf. die Argumente).

```
1 int main()  
2 {  
3   f();  
4   g();  
5 }  
6 void f()  
7 { g(); }  
8 void g()  
9 { i(); }  
10 void h()  
11 { i(); }  
12 void i ()  
13 { }
```

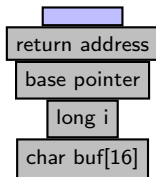


# Basics: stack frame

- ▶ Der stack frame wächst von oben nach unten.
  - ▶ Startet (fast) bei Adresse 0x7fff ffff ffff bei einem 64-bit Linux.
  - ▶ Startet (fast) bei Adresse 0xffff ffff bei einem 32-bit OS.

```
1 void victim(char* arg)
2 {
3     char buf[16] = "abcdefgh";
4     long i;
5     strcpy(buf, arg);
6 }
```

- ▶ Stack frame von victim:



- ▶ In der return address wird gespeichert, wo der Computer fortsetzt, wenn die Funktion `victim` beendet wird.
- ▶ Idee: `strcpy` überschreibt die return address.

# Illustration

```
1 void victim(char* arg)
2 {
3     char buf[16] = "abcdefgh";
4     long i;
5     strcpy(buf, arg);
6 }
7 void payload()
8 {
9     // Do something
10 }
11 void attack()
12 {
13     char buf[1024] = "blablablablablablablabla";
14     ((void**)buf)[5] = &payload; //The return address
15     victim(buf); // Calls payload() when victim returns.
16 }
```

Demo...