



# Dokumentation

## Anwendungssicherheit Praktikum SS2013

Tarek Saier

10.12.2013

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabe 1</b>	<b>1</b>
1.1	Vorgehensweise . . . . .	1
1.2	Beobachtungen . . . . .	1
<b>2</b>	<b>Aufgabe 2</b>	<b>6</b>
2.1	src_audit_1.c . . . . .	6
2.1.1	Funktionsweise des Codeausschnitts . . . . .	6
2.1.2	Schwachstelle . . . . .	6
2.2	src_audit_2.c . . . . .	6
2.2.1	Funktionsweise des Codeausschnitts . . . . .	6
2.2.2	Schwachstelle . . . . .	6
2.3	src_audit_3.c . . . . .	6
2.3.1	Funktionsweise des Codeausschnitts . . . . .	6
2.3.2	Schwachstelle . . . . .	7
<b>3</b>	<b>Aufgabe 3</b>	<b>7</b>
<b>4</b>	<b>Aufgabe 3</b>	<b>7</b>
<b>5</b>	<b>Aufgabe 4</b>	<b>7</b>
<b>6</b>	<b>Aufgabe 5</b>	<b>7</b>
<b>7</b>	<b>Aufgabe 6</b>	<b>7</b>

# 1 Aufgabe 1

Testprogramm `a1.c` bzw. `a1.cpp`:

```
int add_five(int x);

int main() {
    int x = 3, res;
    res = add_five(x);
    return 0;
}

int add_five(int x) {
    int add = 5;
    char c[10] = "AAAAAAAAA";
    return x + add;
}
```

## 1.1 Vorgehensweise

**Compiler:** Obiges Programm mit gcc und g++ compiliert, jeweils mit Parameter `-fstack-protector`, `-fno-stack-protector` oder keinem von beiden.

**Debugger:** In gdb einen Breakpoint ans Ende der Methode `add_five` gesetzt, das Programm ausgeführt und den Stack betrachtet, zudem den disassemblierten Code inspiziert. Siehe Screenshot 1 auf Seite 2, Screenshot 2 auf Seite 3, Screenshot 3 auf Seite 4 und Screenshot 4 auf Seite 5.

## 1.2 Beobachtungen

Sowohl bei gcc als auch g++ (auf dem vorliegenden System beide Version 4.4.5-8) entspricht das Verhalten von `-fno-stack-protector` dem Default. Die Position lokaler Variablen von Prozeduren sind bei beiden Compilern gleich, einige Werte auf dem Stack unterscheiden sich dagegen. Der Unterschied zwischen `-fstack-protector` und `-fno-stack-protector` ist in beiden Fällen deutlich erkennbar. Es werden beim Compilieren zusätzliche Maschinenbefehle generiert und der Aufbau des Stack ist verschieden.

Screenshot 1: gcc mit Parameter `stack-protector`

```
student@hacking01:~/prkt/a1$ gcc -g a1.c -o gcc.out -fstack-protector
student@hacking01:~/prkt/a1$ gdb -q gcc.out
Reading symbols from /home/student/prkt/a1/gcc.out...done.
(gdb) b a1.c:12
Breakpoint 1 at 0x8048438: file a1.c, line 12.
(gdb) run

Breakpoint 1, add_five (x=3) at a1.c:12
12      return x + add;
(gdb) x/16x $esp
0xbffff710: 0xb7f8d7a9 0xb7eb93c5 0xbffff728 0x00000005
0xbffff720: 0x4141cfff 0x41414141 0x00414141 0x1b024500
0xbffff730: 0xb7ff1380 0x0804960c 0xbffff768 0x08048401
0xbffff740: 0x00000003 0xb7fccff4 0x08048470 0xbffff768
(gdb) disass add_five
Dump of assembler code for function add_five:
0x0804840c <add_five+0>:    push    ebp
0x0804840d <add_five+1>:    mov     ebp,esp
0x0804840f <add_five+3>:    sub     esp,0x28
0x08048412 <add_five+6>:    mov     eax,gs:0x14
0x08048418 <add_five+12>:   mov     DWORD PTR [ebp-0xc],eax
0x0804841b <add_five+15>:   xor     eax,eax
0x0804841d <add_five+17>:   mov     DWORD PTR [ebp-0x1c],0x5
0x08048424 <add_five+24>:   mov     DWORD PTR [ebp-0x16],0x41414141
0x0804842b <add_five+31>:   mov     DWORD PTR [ebp-0x12],0x41414141
0x08048432 <add_five+38>:   mov     WORD PTR [ebp-0xe],0x41
0x08048438 <add_five+44>:   mov     eax,DWORD PTR [ebp-0x1c]
0x0804843b <add_five+47>:   mov     edx,DWORD PTR [ebp+0x8]
0x0804843e <add_five+50>:   lea     eax,[edx+eax*1]
0x08048441 <add_five+53>:   mov     edx,DWORD PTR [ebp-0xc]
0x08048444 <add_five+56>:   xor     edx,DWORD PTR gs:0x14
0x0804844b <add_five+63>:   je      0x8048452 <add_five+70>
0x0804844d <add_five+65>:   call   0x8048320 <__stack_chk_fail@plt>
0x08048452 <add_five+70>:   leave
0x08048453 <add_five+71>:   ret
End of assembler dump.
(gdb) █
```

Screenshot 2: gcc mit Parameter no-stack-protector

```

student@hacking01:~/prkt/a1$ gcc -g a1.c -o gcc.out -fno-stack-protector
student@hacking01:~/prkt/a1$ gdb -q gcc.out
Reading symbols from /home/student/prkt/a1/gcc.out...done.
(gdb) b a1.c:12
Breakpoint 1 at 0x80483dd: file a1.c, line 12.
(gdb) run

Breakpoint 1, add_five (x=3) at a1.c:12
12          return x + add;
(gdb) x/16x $esp
0xbffff728: 0x4141f738      0x41414141      0x00414141      0x00000005
0xbffff738: 0xbffff768      0x080483b1      0x00000003      0xb7fccff4
0xbffff748: 0x08048400      0xbffff768      0xb7eb95c5      0xb7ff1380
0xbffff758: 0x00000003      0xb7fccff4      0x08048400      0x00000000
(gdb) disass add_five
Dump of assembler code for function add_five:
0x080483bc <add_five+0>:      push    ebp
0x080483bd <add_five+1>:      mov     ebp,esp
0x080483bf <add_five+3>:      sub     esp,0x10
0x080483c2 <add_five+6>:      mov     DWORD PTR [ebp-0x4],0x5
0x080483c9 <add_five+13>:     mov     DWORD PTR [ebp-0xe],0x41414141
0x080483d0 <add_five+20>:     mov     DWORD PTR [ebp-0xa],0x41414141
0x080483d7 <add_five+27>:     mov     WORD PTR [ebp-0x6],0x41
0x080483dd <add_five+33>:     mov     eax,DWORD PTR [ebp-0x4]
0x080483e0 <add_five+36>:     mov     edx,DWORD PTR [ebp+0x8]
0x080483e3 <add_five+39>:     lea     eax,[edx+eax*1]
0x080483e6 <add_five+42>:     leave
0x080483e7 <add_five+43>:     ret
End of assembler dump.
(gdb) 

```

Screenshot 3: g++ mit Parameter stack-protector

```

student@hacking01:~/prkt/a1$ g++ -g a1.cpp -o g++.out -fstack-protector
student@hacking01:~/prkt/a1$ gdb -q g++.out
Reading symbols from /home/student/prkt/a1/g++.out...done.
(gdb) b a1.cpp:12
Breakpoint 1 at 0x8048528: file a1.cpp, line 12.
(gdb) run

Breakpoint 1, add_five (x=3) at a1.cpp:12
12      return x + add;
(gdb) x/16x $esp
0xbffff710: 0xb7d7f5c5 0xb7d7f3c5 0xb7fcb69c 0x00000005
0xbffff720: 0x41413304 0x41414141 0x00414141 0x736d3c00
0xbffff730: 0xb7ff1380 0x08049784 0xbffff768 0x080484f1
0xbffff740: 0x00000003 0xb7e92ff4 0x08048560 0xbffff768
(gdb) disass add_five
Dump of assembler code for function _Z8add_fivei:
0x080484fc <_Z8add_fivei+0>: push    ebp
0x080484fd <_Z8add_fivei+1>: mov     ebp,esp
0x080484ff <_Z8add_fivei+3>: sub     esp,0x28
0x08048502 <_Z8add_fivei+6>: mov     eax,gs:0x14
0x08048508 <_Z8add_fivei+12>: mov     DWORD PTR [ebp-0xc],eax
0x0804850b <_Z8add_fivei+15>: xor     eax,eax
0x0804850d <_Z8add_fivei+17>: mov     DWORD PTR [ebp-0x1c],0x5
0x08048514 <_Z8add_fivei+24>: mov     DWORD PTR [ebp-0x16],0x41414141
0x0804851b <_Z8add_fivei+31>: mov     DWORD PTR [ebp-0x12],0x41414141
0x08048522 <_Z8add_fivei+38>: mov     WORD PTR [ebp-0xe],0x41
0x08048528 <_Z8add_fivei+44>: mov     eax,DWORD PTR [ebp-0x1c]
0x0804852b <_Z8add_fivei+47>: mov     edx,DWORD PTR [ebp+0x8]
0x0804852e <_Z8add_fivei+50>: lea     eax,[edx+eax*1]
0x08048531 <_Z8add_fivei+53>: mov     edx,DWORD PTR [ebp-0xc]
0x08048534 <_Z8add_fivei+56>: xor     edx,DWORD PTR gs:0x14
0x0804853b <_Z8add_fivei+63>: je      0x8048542 <_Z8add_fivei+70>
0x0804853d <_Z8add_fivei+65>: call    0x80483f4 <__stack_chk_fail@plt>
0x08048542 <_Z8add_fivei+70>: leave
0x08048543 <_Z8add_fivei+71>: ret
End of assembler dump.
(gdb) █

```

Screenshot 4: g++ mit Parameter no-stack-protector

```
student@hacking01:~/prkt/a1$ g++ -g a1.cpp -o g++.out -fno-stack-protector
student@hacking01:~/prkt/a1$ gdb -q g++.out
Reading symbols from /home/student/prkt/a1/g++.out...done.
(gdb) b a1.cpp:12
Breakpoint 1 at 0x80484bd: file a1.cpp, line 12.
(gdb) run

Breakpoint 1, add_five (x=3) at a1.cpp:12
12          return x + add;
(gdb) x/16x $esp
0xbffff728: 0x4141f738      0x41414141      0x00414141      0x00000005
0xbffff738: 0xbffff768      0x08048491      0x00000003      0xb7e92ff4
0xbffff748: 0x080484e0      0xbffff768      0xb7d7f5c5      0xb7ff1380
0xbffff758: 0x00000003      0xb7e92ff4      0x080484e0      0x00000000
(gdb) disass add_five
Dump of assembler code for function _Z8add_fivei:
0x0804849c <_Z8add_fivei+0>:  push    ebp
0x0804849d <_Z8add_fivei+1>:  mov     ebp,esp
0x0804849f <_Z8add_fivei+3>:  sub     esp,0x10
0x080484a2 <_Z8add_fivei+6>:  mov     DWORD PTR [ebp-0x4],0x5
0x080484a9 <_Z8add_fivei+13>: mov     DWORD PTR [ebp-0xe],0x41414141
0x080484b0 <_Z8add_fivei+20>: mov     DWORD PTR [ebp-0xa],0x41414141
0x080484b7 <_Z8add_fivei+27>: mov     WORD PTR [ebp-0x6],0x41
0x080484bd <_Z8add_fivei+33>: mov     eax,DWORD PTR [ebp-0x4]
0x080484c0 <_Z8add_fivei+36>: mov     edx,DWORD PTR [ebp+0x8]
0x080484c3 <_Z8add_fivei+39>: lea     eax,[edx+eax*1]
0x080484c6 <_Z8add_fivei+42>: leave
0x080484c7 <_Z8add_fivei+43>: ret
End of assembler dump.
(gdb) █
```

## 2 Aufgabe 2

### 2.1 src\_audit\_1.c

#### 2.1.1 Funktionsweise des Codeausschnitts

Die Funktion `ownme1` nimmt zwei char-Pointer (`*in` und `*out`) für Strings und ein `size_t maxout` entgegen. Die Funktion bricht ab, wenn `strlen(in)` größer ist als `maxout` oder `*in` nicht mit `#` beginnt. Für jeden weiteren Wert in `*in` ungleich 0 wird:

- `$`: kein weiteres Zeichen in `*out` geschrieben
- `^`: 15 mal "S\_" in `*out` geschrieben
- sonst: ein Zeichen von `*in` in `*out` geschrieben

Am Ende wird die Zahl der geschriebenen Zeichen zurückgegeben.

#### 2.1.2 Schwachstelle

`maxout` wird nur gegen `strlen(in)` geprüft. Für jedes `^` in `*in` wird `*out` aber mehrmals inkrementiert. Im vorherigen Aufruf von `strlen` zählt jedes `^` in `*in` dagegen nur 1. Dadurch lässt sich über die Grenzen von `*out` hinaus schreiben.

### 2.2 src\_audit\_2.c

#### 2.2.1 Funktionsweise des Codeausschnitts

Die Funktion `ownme2` nimmt einen char-Pointer `*input` entgegen. Die ersten 4 Zeichen werden als `size_t size` interpretiert und für den char-Pointer `ptr` `size+1` Byte allokiert. Am Ende werden `strlen(input)` Zeichen aus `input` nach `ptr` kopiert und letztgenannter zurückgegeben.

#### 2.2.2 Schwachstelle

Größe von `ptr` liegt komplett in der Hand der aufrufenden Routine, Außerdem wird beim zweiten `memcpy` die komplette Größe von `input` verwendet und die ersten vier Zeichen nicht abgezogen.

### 2.3 src\_audit\_3.c

#### 2.3.1 Funktionsweise des Codeausschnitts

Die Funktion `ownme3` nimmt ein Feld von `infoz` Strukturen und ein `size_t count`. In einer Schleife werden `count` `infoz` vom `infoz`-Feld `in` in `out` kopiert. Am Ende wird der `infoz`-Zeiger `storage`, welcher auf `out` zeigt, "weggespeicher" und die Anzahl der kopierten `infoz` zurückgegeben.

#### 2.3.2 Schwachstelle

Beim Kopieren der `infoz` wird die Validität der Strukturen nicht überprüft, sondern einfach mit `memcpy sizeof(struct infoz)` große Blöcke kopiert. Daher können die Struktur-Member `setting` und `userid` von `name` aus überschrieben werden.

Außerdem kann `count` größer gewählt werden als die Länge von `input`, sodass weitere Speicherbereiche mit irgendwelchen anderen Inhalten in `out` geschrieben werden.



**3 Aufgabe 3**

**4 Aufgabe 3**

**5 Aufgabe 4**

**6 Aufgabe 5**

**7 Aufgabe 6**