System and Web Security - Homework 1

Marcel Zeller - 3294457
Hugo Selle - 3367876
Ziyuan Huang - 3367724

Problem 1

```
(gdb) disassemble main
Dump of assembler code for function main:
0x08048344 <main+0>:    lea    0x4(%esp),%ecx
0x08048348 <main+4>:    and    $0xfffffff0,%esp
0x0804834b <main+7>:    pushl  -0x4(%ecx)
0x0804834e <main+10>:   push   %ebp
0x0804834f <main+11>:   mov    %esp,%ebp
0x08048351 <main+13>:   push   %ecx
0x08048352 <main+14>:   sub    $0x10,%esp
0x08048355 <main+17>:   movl   $0xef,-0x10(%ebp)
0x0804835c <main+24>:   movl   $0x3,-0xc(%ebp)
0x08048363 <main+31>:   mov    -0xc(%ebp),%eax
0x08048366 <main+34>:   add    -0x10(%ebp),%eax
0x08048369 <main+37>:   mov    %eax,-0x8(%ebp)
0x0804836c <main+40>:   mov    $0x0,%eax
0x08048371 <main+45>:   add    $0x10,%esp
0x08048374 <main+48>:   pop    %ecx
0x08048375 <main+49>:   pop    %ebp
0x08048376 <main+50>:   lea    -0x4(%ecx),%esp
0x08048379 <main+53>:   ret
End of assembler dump.
(gdb) _
```

Zeile 5: **<main+17>** : Kopiere den Wert 0xEF = 239 = 90 + 72 + 77
Zeile 6: **<main+24>** : Kopiere den Wert 3
Zeile 7: **<main+31>** : Kopiere den Wert von der ersten Adresse zu **eax**
<main+34> : Füge den Wert von der ersten Adresse zu **eax** hinzu
<main+37> : Kopiere den Wert von **eax** zu der zweiten Adresse

Problem 2

Main Function:

lea(0, 40): Kopiere Adresse von **esp+4** bzw. **ecx-4** nach **ecx** bzw. **esp**
and(4): Bitweises verknüpfen
pushl(7): Auf Stack schieben (long-Wert)
push(10, 13): Auf Stack schieben
mov(11, 24, 27): Kopiere Wert von erster Adresse zu zweiter Adresse
sub(14): Ziehe den Wert 0x14 von dem Wert der Adresse **esp** ab
movl(17): Kopiere Wert von erster Adresse zu zweiter Adresse (long)
call(30): Rufe myfunction auf
add(35): Füge den Wert 0x14 zu dem Wert der Adresse **esp** hinzu
pop(38, 39): Nächsten Eintrag vom Stack entfernen und in Register schreiben
ret(43): return

```
(gdb) disassemble main
Dump of assembler code for function main:
0x080483b6 <main+0>:    lea    0x4(%esp),%ecx
0x080483ba <main+4>:    and    $0xfffffff0,%esp
0x080483bd <main+7>:    pushl  -0x4(%ecx)
0x080483c0 <main+10>:   push   %ebp
0x080483c1 <main+11>:   mov    %esp,%ebp
0x080483c3 <main+13>:   push   %ecx
0x080483c4 <main+14>:   sub    $0x14,%esp
0x080483c7 <main+17>:   movl   $0x0,-0x8(%ebp)
0x080483ce <main+24>:   mov    -0x8(%ebp),%eax
0x080483d1 <main+27>:   mov    %eax,(%esp)
0x080483d4 <main+30>:   call   0x8048374 <myfunction>
0x080483d9 <main+35>:   add    $0x14,%esp
0x080483dc <main+38>:   pop    %ecx
0x080483dd <main+39>:   pop    %ebp
0x080483de <main+40>:   lea    -0x4(%ecx),%esp
0x080483e1 <main+43>:   ret
End of assembler dump.
(gdb) _
```

Disassembled Code für main Funktion

myfunction:

`push`(0): Auf Stack schieben
`mov`(1, 31...48): Kopiere Wert von erster Adresse zu zweiter Adresse
`sub`(3): Ziehe den Wert 0x28 von dem Wert der Adresse **`esp`** ab
`movl`(6,13,20,59): Kopiere Wert von erster Adresse zu zweiter Adresse (long-Wert)
`subl`(27): Ziehe den Wert 0x13 von dem Wert der Adresse (**`esp-4)`** ab
`call`(59): Rufe **`printf`** auf
`leave`(64):
`ret`(65): return

```
(gdb) disassemble myfunction
Dump of assembler code for function myfunction:
0x08048374 <myfunction+0>:      push   %ebp
0x08048375 <myfunction+1>:      mov    %esp,%ebp
0x08048377 <myfunction+3>:      sub    $0x28,%esp
0x0804837a <myfunction+6>:      movl   $0x2a,-0x4(%ebp)
0x08048381 <myfunction+13>:     movl   $0x80484b0,-0xc(%ebp)
0x08048388 <myfunction+20>:     movl   $0x80484b4,-0x8(%ebp)
0x0804838f <myfunction+27>:     subl   $0x13,-0x4(%ebp)
0x08048393 <myfunction+31>:     mov    -0xc(%ebp),%eax
0x08048396 <myfunction+34>:     mov    %eax,0xc(%esp)
0x0804839a <myfunction+38>:     mov    0x8(%ebp),%eax
0x0804839d <myfunction+41>:     mov    %eax,0x8(%esp)
0x080483a1 <myfunction+45>:     mov    -0x4(%ebp),%eax
0x080483a4 <myfunction+48>:     mov    %eax,0x4(%esp)
0x080483a8 <myfunction+52>:     movl   $0x80484b8,(%esp)
0x080483af <myfunction+59>:     call   0x80482d8 <printf@plt>
0x080483b4 <myfunction+64>:     leave
0x080483b5 <myfunction+65>:     ret
End of assembler dump.
(gdb) _
```

Disassembled Code für myfunction

Problem 3

siehe problem3.c

Es wird 23 in b geschrieben. Dann wird von diesem Wert 15 abgezogen und in a geschrieben. a wird ausgegeben. a und b sind beides Integervariablen.

Problem 4

```
0x08048361 <main+10>:   push   %ebp
0x08048362 <main+11>:   mov    %esp,%ebp
0x08048364 <main+13>:   push   %ecx
0x08048365 <main+14>:   sub    $0x18,%esp
0x08048368 <main+17>:   movl   $0x2a,-0x10(%ebp)
0x0804836f <main+24>:   lea    -0x10(%ebp),%eax
0x08048372 <main+27>:   mov    %eax,0x4(%esp)
0x08048376 <main+31>:   movl   $0x17,(%esp)
0x0804837d <main+38>:   call   0x8048344 <do_something>
0x08048382 <main+43>:   add    $0x18,%esp
0x08048385 <main+46>:   pop    %ecx
0x08048386 <main+47>:   pop    %ebp
0x08048387 <main+48>:   lea    -0x4(%ecx),%esp
0x0804838a <main+51>:   ret
End of assembler dump.
(gdb) info frame
Stack level 0, frame at 0xbffff86c:
 eip = 0x804834a in do_something (c-stack-example.c:5); saved eip 0x8048382
 called by frame at 0xbffff890
 source language c.
 Arglist at 0xbffff864, args: a=23, b=0xbffff878
 Locals at 0xbffff864, Previous frame's sp is 0xbffff86c
 Saved registers:
  ebp at 0xbffff864, eip at 0xbffff868
(gdb) _
```

Zeile für Zeile lesen. c, d, f und g sind nicht initialisiert

0x	ae(c)	81(c)	ef(c)	bf(c)
0x	19(d)	db(d)	f9(d)	b7(d)
0x	6c	f8	ff	bf
0x	82	83	04	08
0x	17	00	00	00
0x	78	f8	ff	bf
0x	2a	00	00	00
0x	b9(f)	83(f)	04(f)	08(f)
0x	f0(g)	1d(g)	ff(g)	b7(g)

nicht initialisiert
initialisiert

```
(gdb) frame 0
#0  do_something (a=23, b=0xbffff848) at problem4.c:5
5           c = a + *b;
(gdb) info locals
c = -1209040466
d = -1208362215
(gdb) frame 1
#1  0x08048382 in main () at problem4.c:15
15          do_something(23, &e);
(gdb) info locals
e = 42
f = 134513593
g = -1208017424
```