Mock exam

Wees duidelijk, en kort maar krachtig in je antwoorden. Je mag gewoon in het Nederlands antwoorden. Succes!

1. (2 points) What would you expect the value of `y` to be directly after the code below, if the underlying hardware uses the 2-complement representation? Motivate your answer.

```
signed char x = -128;
unsigned char y = x;
```

2. (4 points) Consider the code fragment

```
void f(){
  char b,c,d;
  int i;
  unsigned char e;
  ...
}
```

   (a) A compiler is free to choose the order in which the 5 variables are allocated on the stack. What would be a reason for the compiler to swap the order of `e` and `i` in the allocation of these variables on the stack, compared to the order they are declared in the program code?

   (b) Write some code that can be inserted in place of `...` to find out if the compiler did swap the allocation order of `e` and `i`. The code should print 'verwisseld' if the order has been changed, and 'niet verwisseld' if it has not been. Include a short explanation of the idea behind your code.

3. (6 points) Consider the following code

```
#include <stdio.h>
#include <stdint.h>

int main(void)
{
  unsigned char c[4];
  c[0] = 11;
  c[1] = 22;
  c[2] = 33;
  c[3] = (1<<7);

  printf("%p\n", c);
  printf("%p\n", &c);
  printf("%p\n", c+2);
  printf("%p\n", &c+2);
  printf("%d\n", *c);
  printf("%d\n", c[3]+*(c+3));
  return 0;
}
```

Assume that the first call to printf prints `0x7fffb3cc3b20`. What do the other 5 calls to printf print?

4. (6 points) Consider the code below

```
1.  silly(){
2.    char *s = malloc(20);
3.    char *t = malloc(20);
4.    char z[20];
5.    char *u = s;
6.    u[19] = 'a';
7.    free(u);
8.    free(z);
9.  }
```

Explain the 3 different errors in this code. (One of the errors occurs twice, so there are in fact 4 errors.)

5. (4 points) Consider the code below

```
int a[30];
char* c = (char*)a;
long* p = (long*)a;
...
*(c+1) = 'a';          // (1)
*(p+2) = 12345678;     // (2)
```

Assume that an `int` is 4 bytes long and a `long` 8 bytes.

(a) Which entries of the array `a` are affected by assignment (1)? (In other words, for which values of `i` is `a[i]` affected by (1)?).

(b) Which entries of the array `a` are affected by assignment (2)?

Motivate your answers!

6. (4 points) Consider the code below

```
1.   main() { f(); }
2.
3.   f() { int i = 5;
4.         g();
5.         println("%i \n", i);
6.   }
7.
8.   g() { int j = 15;
9.         hackable();
10.        println("%i \n", j);
11.   }
```

Suppose the omitted function `hackable()` contains buffer overflow weaknesses that allow an attacker to corrupt the stack.

(a) Suppose in the call to `hackable()` an attacker overwrites the return address in `f`'s stack frame with the return address in `g`'s stack frame. What will happen?

(b) Suppose in the call to `hackable()` an attacker overwrites the frame pointer in `f`'s stack frame with the frame pointer in `g`'s stack frame. What will happen?

For both questions, assume that there are no mechanisms that will prevent or detect the attacks, and assume that the corrupted stack will not cause runtime problems, so that upon return from a function the execution will continue with the frame and return address retrieved from the stack without any problem.

7. (4 points)

   (a) Explain why on modern machines it is no longer possible for an attacker to execute his own shell code that he manages to insert this into a stack-allocated buffer.

   (b) If the attacker cannot execute his own shell code, how can an attacker still exploit a buffer overflow to effectively do anything that he wants?

8. (6 points) Assume that the following code is running on a server, i.e., input and output is sent through network sockets. Furthermore assume that the server has no memory-protection mechanisms enabled.

```
1.  #include <stdio.h>
2.
3.   void echostr(void) {
4.     char buffer[80];
5.     gets(buffer);
6.     printf(buffer);
7.     printf("\n");
8.   }
9.
10.  int main(void) {
11.    int ret = 1;
12.    while (1) echostr();
13.    return 0;
14.  }
```

   (a) Describe the vulnerabilities in this code.

   (b) Explain what you would do to remotely exploit these vulnerabilities to obtain a shell on the server. You can assume that you are given a 30-byte shellcode. In particular explain

   - what you do to figure out about the stack layout of the process,
   - what you expect to see on the stack,
   - how the stack layout influences the "attack string" (i.e., the data you send to the server to exploit the vulnerability),
   - how the attack string is constructed (what other parts does it contain aside from the shellcode), and
   - how you make sure that the server does not terminate the connection after receiving the attack string.