Tentamen Hacking in C, NWI-IPC025, April 6 2017, 8:30-11:30

Wees duidelijk en kort maar krachtig in je antwoorden.
Je mag gewoon in het Nederlands antwoorden. Succes! (40 points total)

1. (6 points)

   (a) What should be filled in for the `...` below to declare `x`, `y`, `z`, and `w` with the right types?

   ```
   void f(){
    int* i;
    int* a[5];
    ... x = &i;
    ... y = *i;
    ... z = a[3];
    ... w = a;
   }
   ```

   (b) In the code sample above, will `a` be allocated on the heap or the stack?

   (c) What are the values of `i` and `s` after executing the code below? Assume that `p`, `q`, `r` and `s`, have been declared with the correct types.

   ```
   int32_t i = 5;  // Recall: int32_t is the type of signed 32 bit integers
   p = &i;
   q = &p;
   r = *q;
   s = **q;
   (*p)++;
   (*q)++;
   (*r)++;
   s++;
   ```

2. (5 points) Suppose the code below is compiled for a 64 bit architecture.

   ```
   void f(){
     int32_t a[4]; // Recall: int32_t is the type of signed 32 bit integers
     char c;
     int32_t d[2];
     char b[7];
     int64_t i;    // Recall: int62_t is the type of signed 64 bit integers
     ...
   }
   ```

   (a) What would be an obvious way for a compiler to minimise the amount of stack space needed for this function, without compromising execution speed?

   (b) Write a piece of C code that could go in the place of the dots that would return "yes" if the compiler has optimised the stack layout as you suggested in a). Explain the idea behind the code.

3. (5 points)

Consider the following code

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

int main() {
  int32_t x[4];
  x[0] = -2;
  x[1] = 34;
  x[2] = 1|2|4;  //   | is bitwise OR

  printf("%lx \n", x);
  printf("%lx \n", x+2);         // (a)
  printf("%lx \n", &x);          // (b)
  printf("%lx \n", &x+2);        // (c)
  printf("%lx \n", *(x+1) & 2);  // (d)    & is bitwise AND
  printf("%lx \n", *x + x[2]);   // (e)
  return 0;
}
```

Recall that `%lx` prints a long in hexadecimal notation. Assume that the target machine uses two's complement to represent negative integers.

If the first call to `printf` prints `abcdabcdef00`, what do the other calls to `printf` print?

4. (8 points)

Consider the code below. The code is legal C.

```
 1.   char* f(char* w) {
 2.     char *groet = "hello";
 3.     char *t = malloc(50);
 4.     char *u = malloc(50);
 5      char z[50];
 6.     strcpy(u,groet); // copies the string groet to u
 7.     u[5] = '!';
 8.     t = w;
 9.     printf("String t is now %s.\n", t);
10.     t = u;
11.     printf("String t is now %s.\n", t);
12.     free(t);
13.     free(z);
14.     return u;
15.  }
```

(a) What are the 5 errors in the program above? (One error occurs two times, so there are in fact 6 errors.) For each error, mention the line number(s) involved, and explain what is wrong.

(b) Could the print statement in line 9 cause a segmentation fault? Motivate your answer.

(c) Is it possible to say what the `printf` statement in line 11 will print (if the program does not crash before)? If so, say what it prints; if not, explain why.

5. (8 points) Consider the code below. We assume the compiler does not optimise away redundant variables such as `secret`. The code for functions `f` and `g` is not shown.

```
 1.  main () {
 2.    proc1();
 3.  }
 4.
 5.  proc1() {
 6.     int secret = 1234;
 7.     proc2();
 8.  }
 9.
10.  proc2() {
11.     int public = 024;
12.     f();
13.     printf("f is done");
14.     g();
15.     printf("The area code is %i", public);
16.  }
```

Suppose that the function `f` contains some memory weaknesses, which an attacker can exploit to inspect and/or corrupt the stack by supplying malicious input. We assume there are no countermeasures against this, such as stack canaries or a non-executable stack.

In answering the questions below, when talking about stack frames always make it clear to which function these belong.

(a) Could an attacker corrupt the stack during the call to `f` in such a way that after returning from `f` the print statement on line 13 is not executed, but in all other respects execution continues as it normally would?

Motivate your answer, by explaining why not, or by explaining how the attacker might achieve this.

(b) Could an attacker corrupt the stack during the call to `f` so that the value of `secret` will be printed instead of the value of `public` when execution reaches line 15?

Motivate your answer, by explaining why not, or by explaining how the attacker might achieve this.

(c) Would having a non-executable stack prevent any attacks you discussed in your answers to a) or b) above?

(d) To provide some additional security,, the programmer replaces line 12 with the following lines:

```
12a.     int test = 2525;
12b.     f();
12c.     if (test != 2525) { exit(-1);}
```

Does this provide any additional security? Motivate your answer.

(Here assume that the compiler does not optimise the superfluous if-statement away because the `then`-branch is unreachable.)

6. (8 points) The code running on the vulnerable server hackme.cs.ru.nl for the last assignment was something like

```
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.    while (1) { echostr(); }
12.    return 0;
13.  }
```

(a) What are the two security vulnerabilities in this code?

(b) How would you fix these vulnerabilities?

(c) To craft the attack string to launch a shell, in addition to the actual shell code, the attacker needs still some additional information. What is additional information that the attacker needs?

(d) Did the vulnerable server that you had to take over using the given shell code run with a non-executable stack? Or was that impossible to tell from the attack? Motivate your answer.