Grenoble INP | **UGA**
Master CybserSecurity & Master MoSiG | **Year 2017-18**

Software Security & Secure Programming

Written Assignment - Tuesday November the 14th, 2017

Duration : 60 minutes – **Authorized documents :** one A4 sheet of paper

Exercise 1. (∼ 10 pts) We consider the following C code :

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char *alloc_and_copy(char *dst, char src[], unsigned int nbcells) {
    unsigned char size;
    size = nbcells;
    dst = (int *) malloc(size);
    strcpy(dst, src);  // copy src into dst
    return dst ;
}

int main(){
    char t1[256];
    char *t2;
    scanf("%s", t1);    // read the content of t1 from the keyboard
    t2=alloc_and_copy(t2, t1, 256);
    sprintf("%s", t2);    // print the content of t2 on the screen
    free(t2);
    return 0;
}
```

Q1. Find the vulnerabilities in this code (you should find at least 3 vulnerabilities, may be more!)

Q2. For each of them explain why it is a vulnerability, and if (and how) this vulnerability could be exploited by a malicious user (indicating which gain this attacker would get).

Q3. Update this code to make it secure (while preserving the same “nominal behavior”).

Q4. Would each of these vulnerabilities occur in a “secure” programming language like Rust[1]? Explain why, or why not . . .

(to be continued on the next page)

1. or Java or Ada, etc.

Exercise 2. (∼ 5 pts)

According to the CERT, in a C program, *"a simple yet effective way to eliminate dangling pointers and avoid many memory-related vulnerabilities is to set pointers to NULL after they are freed"* (rule MEM01-C). Explain, by giving a suitable example for each cases,

1. why this recommandation may help to make **some** *use-after-free* **non exploitable**, preventing an attacker to break confidentiality or integrity properties ;
2. why it is not a **sufficient condition** in general (some exploitable use-after-free may still occur).

Exercise 3. (∼ 5 pts)

The following C code initializes a **secret** key and prints later on a **public** directory name. Running these code (several times) an observer is able to get some information on the key.

```c
#include <stdio.h>
#include <string.h>

void foo (char k[], int size) {
  int i;
  for (i=0; i<size; i++)
       k[i] = k[i] + 1 ;
}

void main() {
  char key[1024]; // secret key
  int size;       // secret key size
  char buff[1024];
  char dir[1024];  // public dirname

  init(&key, &size); // initialize the secret key and its size
  strncpy(buff, key, size); // copy size bytes of key into buff
  if (buff[0]== 0xFF) {
      strcpy(dir, "./tmp");
  } else {
      foo(key, size);
      strcpy(dir, ".");
  } ;
  printf("%s\n", dir);
}
```

Indicate :

1. which are the instructions leaking some secret informations on the key ?
2. which precise information on the key can be actually retrieved ?