Grenoble INP **UGA**
Master CyberSecurity **Year 2023-24**

Software Security & Secure Programming

Written Assignment - Tuesday November the 14th, 2023

Duration: 1h15 – Answers can be written either in English or in French – All documents allowed.

Exercise 1 (∼ 6 pts)

We consider the two (independent) functions `f1` end `f2` given below. We assume that in both cases, the string parameter `s[]` is controled by the user.

```
1 #define L 256
2
3 int f(char s[]) {
4   char buf[L] ;
5   for (int i=0 ; i<strlen(s) ; i++) {
6         buf[i]=s[i] ;
7   } ;
8   printf("%s", s) ;
9 }
```

```
1 #define L 256
2
3 void f(int s[]) {
4   int x ;
5   for (int i=0 ; i<L ; i++) {
6         scanf("%d", &x) ;
7         if (i%2) s[i] = x ;
8   } ;
9 }
```

Q1. We assume first that these functions are compiled and executed without any specific protections. Tell, fo each of them, if the function contains a vulnerability that could be exploited by an attacker, indicating the gain it may obtain (if any). You should explain your answer in a few line and/or with the help of a figure.

Q2. We now assume that these functions are compiled with the `-fstack-protector` flag, to include *stack canaries*. Does it modify the answers you gave for `Q1`? If a function is still vulnerable, which protection mechanism would ou advise to prevent its exploitability? Explain your answers ...

Exercise 2 (∼ 7 pts)

When compiling (without any specific option) and running the C program given on the next page we obtain the value `0` printed on the screen.

Q1. Explain why, preferably with the help of a figure.

Q2. What are the implications of this behavior with respect to security? Give a small code example, using the same "vulnerable pattern", and allowing an attacker to break some expected security property.

Q3. Compiling this program with Address Sanitizer[1] does not change its behavior (no error detected, still prints `0`). Propose and discuss some techniques that would (help to) detect this vulnerability:

1. at compile-time;
2. at run-time.

[1] using the flag `-fsanitize=address`

```
1  int *foo() {
2   int a ;
3   int *x ;
4   a = 42 ;
5   x = &a ;
6   return x ;
7  }
8
9  int bar() {
10  int buf[20]  ;
11  for (int i=0;i<20;i++)
12   buf[i]=0 ;
13 }
14
15 int main() {
16  int *p ;
17  p = foo() ;
18  bar() ;
19  printf("%d\n", *p) ;
20  return 0 ;
21 }
```

Exercise 3 (∼ 7 pts)

We consider the function `foo` below:

```
1  void foo(char s[]) {
2    char buf[L];
3    strcpy(buf,s);
4  }
```

We know that a remote server, owned by a criminal organisation[2], executes an application calling function `foo` and, on this server:

- the size of the string parameter `s` can be up to 300 bytes long, and we do control its content remotely;
- buffer `buf` lies in the stack at address `0xFFDEAD00`[3];
- the size of the buffer (`L`) is between 40 an 100 bytes (we do not know its exact value);
- the distance between the end of the buffer and the return address location of function `foo` in the stack is 8;

We also have a 30 bytes long sequence SC (shell-code) which, when executed on the remote server, would allow to crash it definitely (which is our goal!).

Give a possible content for string `s` in order to activate our shell-code SC when function `foo` terminates. Remember that in C language the null-byte `0x00` is a string termination mark, and beware, only one try is allowed, and it should be successful!

Here again you can explain your solution with a picture ....

[2] the one you wants!
[3] addresses are 4 bytes long on this machine