Université Grenoble Alpes **Grenoble INP**
UFR IM2AG **ENSIMAG**

Master 2 CyberSecurity
Software Security and Secure Programming

Exercices on Access Control and Information Flow

Exercise 1

Let consider the following code, where security classes are ordered S > C > U (constant values being in class U):

```
x : integer class S;
y,z : integer class C;
t : integer class U;

y := 2; z:= 3;
x := y+z ;
if ( y<5 ) then
      t := 4;
else
      t := 3;
```

We require that a user of given security class should not get access to information belonging to a higher class.

Q1. Is this program correct for a user of class C ?

Q2. And for a user of class U ?

Exercise 2

Assuming parameters n and k are "high" (confidential), is this function potentially leaking information ? And if yes, where and how ?

```
int crypto_secretbox_open
  (unsigned char *m,  const unsigned char *c,
        unsigned long long clen,
   const unsigned char *n,  const unsigned char *k)
{
 int i;
 unsigned char subkey [32];

 if (clen < 32) return -1;

 subkey = crypto_stream_salsa20(32,n,k);

 if (crypto_auth_hmacsha512_verify(c,c+32,clen -32, subkey)!=0)
      return -1;
 crypto_stream_salsa20_xor(m,c,clen ,n,k);

 for (i = 0;i < 32;++i)
      m[i] = 0;

 return 0;
}
```

Exercise 3

The objective is to identify vulnerable statement able to write untrusted (i.e. user controlled) values into memory. We use the following notation:

- a value is said tainted (T) if it depends on a user input;
- it is said untainted (U) otherwise.

Q0. Explain why/how this taint analysis problem is related to non-interference ?

We consider the following piece of code, assuming that variable x0 is a tainted data and f() is a "dangerous" function which should not be called with a tainted argument.

```
while (i < 10) {
      x3 = x2 ;
      x2 = x1 ;
      x1 = x0 ;
      i = i+1 ;
} ;
f (x3)
```

Q1. Discuss for which initial values of i this code is dangerous or not ...

Exercise 4

We consider the following function:

```
1 void buildfname ( char *gecos , char *login , char * buf)
2 {
3    char *p;
4    char *bp = buf ;
5
6    for (p = gecos ; *p != '\0 ' && *p != ',' && *p != ';' && *p != '%'; p ++){
7    if (*p == '&') {
9                 strcpy (bp , login );
10                *bp = toupper (* bp );
11                while (* bp != '\0 ')
12                      bp ++;
13          } else {
14                bp ++;
15                *bp = *p;
16          }
17   }
18   *bp = '\0 ';
19 }
```

The objective is to identify vulnerable statement able to write *untrusted* (i.e. user controlled) values into memory. We use the following notation:

- a value is said **tainted** (T) if it depends on a user input;
- it is said **untainted** (U) otherwise.

Q0. Explain why/how this *taint analysis* problem is related to *non-interference* ?

Q1. Which instructions perform **memory write** operations (i.e, are potentially vulnerable) ?

Q2. Assuming both parameters `gecos` and `login` are tainted, how does this taint propagate to potentially vulnerable instructions ?

Q3. Same question if only `gecos` is tainted

Q4. Same question if only `login` is tainted

Exercise 5

In some languages like Java the compiler checks if (local) variables are initialized before being used (objects and global variables are initialized by the compiler).

For instance compiling the following programs will fail:

P1 : { x := 3; y:= (x+3); z := (y+z); }
P2 : { x := 3; if (x > 10) then y:=1 ; else z:= 2 ; end ; x:= (y+3); }

Q1. With respect to variable initialization, several solutions can be adpoted depending on the programming language semantics:

1) nothing is done (no verification)
2) uses of uninitialized variable are detected at runtime
3) variables are initialized by the compilers
4) uses of uninitialized variable are detected at compile time

Discuss these different options with respect to:
1) cost
2) consequences from a safety and/or security point of view

Q2. Propose an algorithm allowing to compute at compile time the set of non-initialized variable for a small language (assignment, conditional statement, iteration).