LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Ulf Kargén

Written exam

TDDC90 Software Security

2025-03-20

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ulf Kargén, 013-285876

Instructions and grading

There are 7 questions on the exam. Your grade will depend on the total points you score. The maximum number of points is 38. The following grading scale is preliminary and might be adjusted during grading. You may answer in Swedish or English.

Grade	3	4	5
Points required	19	27	32

Question 1: Secure software development (4 points)

Consider the CORAS method for risk analysis. If you would want to integrate the other two risk analysis methods discussed in the course – *misuse cases* and *attack trees* – into CORAS, in which step of CORAS would each respective method fit best? For each of *misuse cases* and *attack trees*, explain in which step of CORAS it would fit best, and clearly motivate how it would contribute to the CORAS workflow in that step. Only answer with one CORAS step per method.

Question 2: Memory safety (5 points)

a) Explain how *stack pivoting* works, and why it is used.

b) Consider an out-of-bounds read vulnerability, which allows an attacker to read some data past the end of a stack buffer. For each of the following two mitigations, explain whether or not it can mitigate this kind of attack, and why.

   i. ASLR
   ii. CFI

Question 3: Design patterns (3 points)

a) Come up with a (realistic) example of a vulnerability and corresponding attack that is made possible by failing to adhere to the *clear sensitive information* pattern.

b) What is typically the cause of use-after-free (UAF) bugs? Name a design pattern that can help reduce the risk of UAF.

Question 4: Web security (7 points)

a) Explain, using an example, how an XXE vulnerability could be exploited to perform an SSRF attack. Your example should be detailed enough, so that it is possible to understand every principal step of the attack.

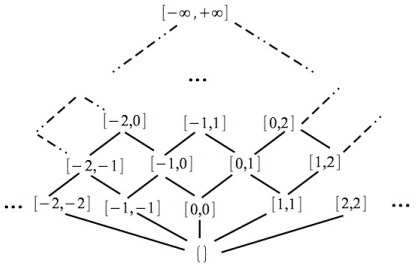
b) Explain how a CSRF attack works.

c) Why can it be a good idea to use the HttpOnly flag when creating cookies in your web app? Clearly motivate your answer.

Question 5: Static analysis (7 points)

Assume `int` denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows). Consider the following procedure.

```
1  int foo(int a, int b){
2    int rslt = 0;
3    int i = 0;
4    while(i <= a){
5      if(b % 3 == 0){
6        rslt = rslt + 3;
7      }else if(b % 3 == 1){
8        rslt = rslt + 2;
9      }else{
10       rslt = rslt + 1;
11     }
12     i = i + 1;
13     b = b + 1;
14   }
15   assert(rslt >= a);
16   return rslt;
17 }
```



Questions:

1. Symbolic execution:
   - Suppose the assertion at line 15 is removed (i.e., commented out). How many possible (i.e. feasible) paths does the method `foo` have if the variable `a` can assume any value in `[-5,5]`? (2 pt)
   - Now suppose the assertion is put back at line 15 (i.e., it is not commented out anymore). Give, without discussing its possible satisfiability, the SSA formula for the path condition that passes once through line 10 (i.e., includes the execution of `rslt = rslt + 1`) and that violates the assertion. Observe the path condition might not be satisfiable, but it should follow the control flow of the program. (1 pt)
2. Abstract interpretation: Annotate, after convergence of a most precise analysis, the start of each line in `foo` with an abstract element associated to each one of the defined variables (including the parameters passed to `foo`). Use, for each variable or parameter, the interval domain, i.e., the abstract elements depicted in the lattice to the right of `foo`. Observe this is not the sign domain discussed in the course. (2 pt)
3. Give, without justification, the weakest condition P such that executing the sequence of three assignments `z= x + y; x= y; y= z - x;` (where `x`, `y` and `z` are integer variables with no integer overflows) from a configuration satisfying P will always result in a condition satisfying the condition ($Q : x + y \geq 0$ and $y \leq x$). In other words, give the weakest (i.e., most general) predicate P such that the Hoare triple $\{P\}$`z= x + y; x= y; y= z - x;`$\{Q\}$ holds (i.e., is always true or valid). (2 pt).

Question 6: Security testing (6 points)

a) Briefly explain two reasons why automated fuzzing of web applications is often harder than fuzzing, e.g., a desktop program written in C.

b) Explain why a Heartbleed-like bug, where it is possible to read a few kB past the end of a buffer, might be hard to find via fuzzing. Also explain how a *sanitizer* (such as AddressSantizer) could help here.

c) Give pseudocode for a small program with a bug that would be easy to find using concolic testing, but hard to find using mutation-based fuzzing. Explain your reasoning.

Question 7: Vulnerabilities in C/C++ programs (6 points)

The code on the next page shows a function that takes a simple greyscale bitmap image as input, and renders it as ASCII art. The code has a serious security bug.

It can be assumed that the input buffer (`data`) always contains exactly `width*height` bytes.

a) Identify the bug in the code, and state what kind of bug it is (out of the security bug types discussed in the course).

b) Clearly explain, using an example, what an input to the function should look like in order to trigger the bug, and what the consequences of a successful exploit could be (i.e., what the bug allows the attacker to do).

c) Show using code/pseudocode (and a clear motivation) how the bug should be fixed.

```c
// Returns a malloc-allocated string with the ASCII-art render of an 8-bit
// grayscale bitmap image. Returns a NULL-pointer in case of error.
char* render_ascii(unsigned int width,
                   unsigned int height,
                   const char* data)
{
   if( (width+1) > UINT_MAX / height)
      return NULL;

   // Allocate memory for pixels + line breaks after rows + NULL terminator
   char* output = malloc( (width+1) * height + 1 );
   size_t out_pos = 0;

   for(unsigned int y = 0; y < height; y++) {
      for(unsigned int x = 0; x < width; x++) {
         unsigned char pixel = data[y*width + x];
         char character;

         if(pixel < 64)
            character = '#';
         else if (pixel < 128)
            character = '*';
         else if (pixel < 192)
            character = '-';
         else
            character = ' ';

         output[out_pos++] = character;
      }
      output[out_pos++] = '\n'; // Add line-break
   }

   // Add NULL-terminator to finish output
   output[out_pos] = 0;

   return output;
}
```