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

Written exam

TDDC90 Software Security

2025-01-18

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)

a) For each of the following techniques or methods, state *without motivation* in which phase of the software development lifecycle it would be most appropriate to apply the technique/method. Only state one phase per item. Each correct answer gives 0.5 points, and each incorrect answer -0.5 points. The minimum amount of points is 0 and the maximum 2.

   i. Static analysis
   ii. Misuse cases
   iii. Fuzzing
   iv. Attack trees

b) In a few sentences, explain the following two concepts:

   i. Attack surface reduction
   ii. Defense in depth

Question 2: Memory safety (7 points)

a) Could ROP be used to exploit a use-after-free bug? If no, explain why it is impossible. If yes, explain how.

b) Clearly explain why DEP and ASLR must always be used together in order to be effective.

c) Under what circumstances is an *integer overflow* a security bug? Give a small (pseudo)code example of an *exploitable* integer overflow vulnerability.

Question 3: Design patterns (2 points)

Explain the intent and motivation for the *Privilege separation* pattern.

Question 4: Web security (6 points)

a) Using pseudo-code, write server-side code that contains a vulnerability that allows for *reflected* XSS. Your code should be detailed enough that it is clear how XSS attacks can be made. Explain your code in English (or Swedish). Give a (realistic) example of how an attacker could use the bug to attack a user of the affected site. Finally, show how the code should be altered in order to mitigate the attack.

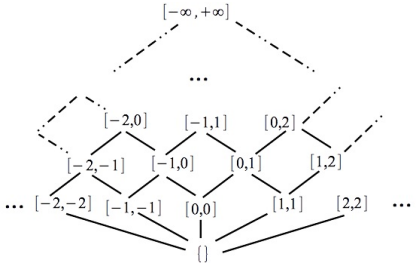
b) Explain what an *Insecure Direct Object Reference* vulnerability is and give an example of this kind of vulnerability.

c) Explain why using *salting* is important when storing passwords for a web app and explain how salting works.

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 % 2 == 0){
6        rslt = rslt + 2;
7      }else{
8        rslt = rslt + 1;
9      }
10     i = i + 1;
11   }
12   assert(rslt >= a);
13   return rslt;
14 }
```



Questions:

1. Symbolic execution:
   - Suppose the assertion at line 12 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 `[-10,10]`? (1 pt)
   - Now suppose the assertion is put back at line 12 (i.e., it is not commented out anymore). Give, without discussing its possible satisfiability, the SSA formula for one path condition that violates the assertion. There might be several path conditions. Just give an SSA formula for one of them. (2 pts)
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 pts)
3. Give, without justification, the weakest condition P such that executing the sequence of three assignments `z= x; x= y; y= z+z;` (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; x= y; y= z + z;`$\{Q\}$ holds (i.e., is always true or valid). (2 pts).

Question 6: Security testing (6 points)

a) Mutation-based fuzzers typically do not work that well for fuzzing implementations of stateful network protocols. Explain why and name a fuzzing technique more suited for this use case.

b) What is the problem with *magic constants* in fuzzing? Which of blackbox, greybox, or whitebox fuzzing is best suited to deal with this problem? Briefly motivate

c) In the context of mutation-based fuzzing, what is a seed input? Why would an empty or random seed input probably not work that well for a black-box mutational fuzzer?

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

The function `add_record` on the next page takes a name and a salary as arguments, and outputs a formatted entry to file. For example, given the name "John Doe" and the salary 4000, the string "`John Doe: $4000`" will be written to file. The input parameter `name` can be assumed to always point to a valid NULL-terminated string, but both the contents of the string as well as the salary may originate from an untrusted source. The function contains at least one serious vulnerability.

a) Identify the bug in the code, name the vulnerability type, and explain how the bug could be triggered. Your answer should include an example of an input that triggers the bug (but you don't need to explain how to craft a full exploit).

b) Explain how to fix the bug.

```c
/* Calculates the number of letters (i.e. digits) that are needed
   to represent a decimal number as an ASCII string */
size_t count_digits(unsigned int number)
{
  unsigned int left = number;
  size_t n = 0;
  while(left != 0) {
    left = left / 10;
    n++;
  }
  return n;
}


void add_record(const char* name, unsigned int salary)
{
  char buffer[256];

  size_t len = strlen(name);
  size_t num_digits = count_digits(salary);

  /* 5 extra bytes required for colon and space after name +
     dollar sign, endline and NULL-terminator */

  if(len > SIZE_MAX - 5 || len + 5 > SIZE_MAX - num_digits) {
    printf("Integer oferflow!\n");
    exit(1); /* exit program with error */
  }

  len = len + num_digits + 5;

  if(len > sizeof(buffer)) {
    printf("Too long string!\n");
    exit(1);  /* exit program with error */
  }

  /* Output formatted string to buffer (in the format string,
     %s denotes a string, and %u denotes an unsigned int that is
     printed as a decimal number) */
  sprintf(buffer, "%s: $%u\n", name, salary);

  // Write buffer to file
  fputs(buffer, global_file_handle);
}
```