Information page for written examinations at Linköping University

Examination date	2019-08-28
Room (1)	TER4(4)
Time	14-18
Edu. code	TDDC90
Module	TEN1
Edu. code name **Module name**	Software Security (Software Security) Written examination (Skriftlig tentamen)
Department	IDA
Number of questions in the examination	7
Teacher responsible/contact person during the exam time	Ulf Kargén
Contact number during the exam time	013-285876
Visit to the examination room approximately	15:00, 17:00
Name and contact details to the course administrator (name + phone nr + mail)	Annelie Almquist, 013-282934, annelie.almquist@liu.se
Equipment permitted	Dictionary (printed, NOT electronic)
Other important information	
Number of exams in the bag	

LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Nahid Shahmehri

Written exam

TDDC90 Software Security

2019-08-28

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ulf Kargén, 013-285876

Instructions and grading

You may answer in Swedish or English.

There are 7 questions on the exam. Your grade will depend on the total points you score. The maximum number of points is 40. The following grading scale is preliminary and might be adjusted during grading.

Grade	**3**	**4**	**5**
Points required	20	29	35

Question 1: Secure software development (4 points)

a) How is probability and consequence represented in CORAS, and how are risks compared? Use a small example to illustrate your explanation.

b) In which phase of the security development life cycle should static analysis be used?

Question 2: Exploits and mitigations (5 points)

Consider the two attack methods below, each designed to overcome a specific exploit mitigation. For each of the two methods, give a high-level explanation of the exploit mitigation technique it was designed to circumvent, and how the attack is able to circumvent that mitigation

a) Return-to-libc

b) Heap spraying

Question 3: Design patterns (5 points)

a) Explain the design pattern *secure chain of responsibility*. Your answer should include a diagram, pseudo-code and an explanation of why and when the pattern should be used.

b) Explain the intent and motivation of the *Resource Acquisition Is Initialization* (RAII) pattern. Name a type of vulnerability that is avoided by using this pattern.

Question 4: Web security (6 points)

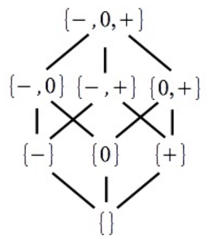
a) Explain, using a combination of pseudo-code and English, why email clients should block images from untrusted sources. Your answer should be a complete example of how an attacker could use images to stage a cross-site request forgery.

b) Using pseudo-code, write server-side code that contains a vulnerability that allows for SQL injections. Your code should be detailed enough that it is clear how SQL injections can be made. Explain your code in English. Give an example of a client-side request that would exploit the vulnerability in your code. Finally, suggest a modification of your code such that the vulnerability is removed. Explain why your mitigation strategy works

Question 5: Static analysis (7 points)

The following function computes the product $n * n$ of a natural number n. Here, `int` denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows).

```
1  int foo(int n){
2    if( (n <= 0))
3       return 0;
4    int i = 0;
5    int r = 0;
6    while (i < n){
7      r = r + 2*i + 1;
8      i = i + 1;
9      assert(r == i * i);
10   }
11   assert(r == n*n);
12   return r;
13  }
```



We aim to check the assertions `(r == i*i)` at line 9 and `(r == n*n)` at line 11.
Questions:

1. Consider first the assertion `(r == n*n)` at line 11:

   (a) Symbolic execution: Give a path formulas that would correspond to taking the else outcome of the if statement (line 2), entering the loop once (i.e., one iteration of the loop), exiting the loop to get to line 11 and violating the assertion there (i.e. violating the `(r = n*n)` assertion). (2 pt)

   (b) Abstract interpretation: can an abstract interpretation analysis based on the sign abstract domain mentioned above establish that the assertion is never violated? explain by annotating each line with the abstract element associated to each variable (i.e., each one of `i`, `r` and `n`) and obtained at the end of such an analysis (i.e., after the analysis converges). (1pt)

2. Consider now the assertion `(r == i*i)` at line 9:

   (a) What does it mean for the predicate $wp(stmt, Q)$ to be the weakest precondition of a predicate Q with respect to a program statement $stmt$? (1 pt)

   (b) Give P_8 defined as the weakest precondition of the predicate `(r == i*i)` with respect to the assignment `i = i + 1` at line 8; then give P_7 defined as the weakest precondition of the predicate P_8 with respect to the assignment `r = r + 2*i + 1` at line 7. (2pt)

   (c) Is P_7 an invariant of the loop? would having P_7 as an invariant of the loop be enough to establish the assertion at line 9? justify. (1pt)

Question 6: Security testing (7 points)

a) In a few sentences each, describe the role of the *fuzzer*, *dispatcher*, and *assessor* in a fuzzing framework.

b) What is the path explosion problem in concolic testing?

c) Explain why detecting cross site request forgery (CSRF) bugs using automated testing is often very difficult.

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);
}
```