Information page for written examinations at Linköping University

Examination date	2015-04-08
Room (1)	TER3
Time	14-18
Course code	TDDC90
Exam code	TEN1
Course name **Exam 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)	Madeleine Häger Dahlqvist, 013-282360, madeleine.hager.dahlqvist@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

2015-04-08

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 this course we mentioned three additional activities that can be requested by security advisors for projects using SDL. Name two of these.

Question 2: Exploits and mitigations (5 points)

a) Name one benefit of using a register trampoline over a NOP-sled when exploiting a stack-based buffer overflow. Motivate your answer!

b) Why may the register trampoline method not always be applicable?

c) Depending on the implementation, ASLR may or may not be able to mitigate register trampoline exploits. What property must an ASLR implementation have in order to prevent such exploits? Briefly explain your answer.

Question 3: Design patterns (5 points)

Explain the following two design patterns: *secure factory* and *secure logger*. For each pattern your answer should include a diagram, pseudo-code and an explanation of why and when the pattern should be used.

Question 4: Web security (6 points)

For each of the two (web)vulnerabilities: *command injection* and *SQL injection*, address the following (in the context of web security).

a) Give a brief example of a possible consequence if an attacker is successful in exploiting the vulnerability.

b) Write an example of server-side code (pseudo-code) that allows for the vulnerability (explain why the code is vulnerable).

c) Give an example of a request to the server-side code that would exploit the vulnerability (i.e. an attack).

d) Give an example of how changes to the code can mitigate the vulnerability so that the attack is no longer effective (explain why the code works).

Question 5: Static analysis (7 points)

Consider the following `foo` function:

```
1  int foo(int x){
2    int y=0;
3    int z=x;
4
5    while (x!=0){
6      if(x > 0){
7        x--;
8        y++;
9      } else {
10       x++;
11       y--;
12     }
13   }
14   assert(y == z);
15   return 1;
16 }
```

We check the assertion `(y == z)` at line 14. For this, we compute the reachable variables values at that line with two different methods:

while *get value* v_x *for* x **do**
  `/* compute effects of lines 1-13 */`
  let v_x, v_y, v_z be values of x, y, z at line 14;
  `/* check assertion */`
  if $v_y \neq v_z$ **then**
    report violated assertion;
  end
end

Method 1: Concrete value based

`/* compute effects of lines 1-4 */`
let both I_x and I_z be the interval [`int_min`, `int_max`];
let I_y be the interval $[0, 0]$;
`/* compute effects of loop, lines 5-13 */`
repeat
  let J_x, J_y, J_z respectively equal I_x, I_y, I_z;
  compute new intervals I_x, I_y, I_z resulting from lines 5-13 ;
until I_x *equals* J_x *and* I_y *equals* J_y *and* I_z *equals* J_z;
`/* check assertion */`
report violation if there are $v_y \in I_y$ and $v_z \in I_z$ with $v_y \neq v_z$;

Method 2: Abstract intervals based

Questions:

a) Which method is sound? Give a drawback and an advantage. (2pt)

b) Which method is complete? Give a drawback and an advantage. (2pt)

c) Give a formula that captures a symbolic execution that corresponds to one iteration of the loop, takes the then branch of the if statement, and violates the assertion. Is the formula satisfiable? (1pt)

d) Can the assertion be violated? Use your own arguments. (2pt)

Question 6: Security testing (7 points)

a) What is the benefit of using a dynamic-analysis tool such as Valgrind or AddressSanitizer during fuzzing? Give an example! What is the downside?

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

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 small C++ function shown on the next page reads text from a file, one line at a time, and concatenates all lines into one string before proceeding to process the concatenated text. (The nature of that processing is not relevant here.) The code contains at least one serious vulnerability, which could potentially be exploited to allow arbitrary code execution.

a) Identify the vulnerability, and explain what the input file should look like in order to trigger the bug. (You *don't* need to explain in detail how to exploit the vulnerability.)

b) Explain, in words *and* pseudocode, how to fix the bug.

You can assume that the comments in the code correctly describe the behaviour of library functions, etc. You don't need any additional knowledge about the library functions used than what is given in the comments.

```
void read_and_process(istream& in_file)
{
   const int MAX_DATA = INT_MAX;
   const int BUFSIZE = 1000;

   // Allocate a buffer on the heap with MAX_DATA bytes.
   char* concatenated = new char[MAX_DATA];
   // 'concatenated' should initially contain an empty string.
   concatenated[0] = 0;
   int total_read = 0;

   char buffer[BUFSIZE];

   // 'getline' reads one line of text (until a line delimiter is reached)
   // from 'in_file' into 'buffer'.
   // A maximum of 'BUFSIZE' bytes is written to 'buffer', including the
   // null terminator.
   // If end-of-file is reached, or the current line contains 'BUFSIZE'
   // characters or more, the call to 'getline' will evaluate to false,
   // and the loop will be terminated.
   while(in_file.getline(buffer, BUFSIZE))
   {
      int len = strlen(buffer) + 1;
      if(total_read + len > MAX_DATA)
      {
         printf("Error: Too much data");
         exit(1); // Quit program
      }

      // Append string in 'buffer' to existing string in 'concatenated',
      // starting from the position of the old null terminator in
      // 'concatenated'.
      strcat(concatenated, buffer);
      total_read = strlen(concatenated);
   }
   // Complete string read. Process it...
   process(concatenated);
}
```