Information page for written examinations at Linköping University

Examination date	2017-08-23
Room (1)	TER2(14)
Time	8-12
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	Marcus Bendtsen
Contact number during the exam time	0733-140708
Visit to the examination room approximately	09:00, 11:00
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

2017-08-23

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Marcus Bendtsen, 0733-140708

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) Explain by example how attack trees are created (come up with a scenario on your own, and make sure that you explain all the details of attack trees).

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) What kinds of benefits, if any, would a register-trampoline attack offer over using a NOP-sled if the vulnerable program is compiled with stack cookies? Motivate your answer.

Question 3: Design patterns (5 points)

Explain the following two design patterns: secure factory and privilege separation. 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)

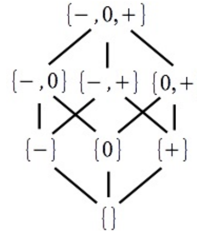
a) Using pseudo-code, write server-side code that contains a vulnerability that allows for SQL injections. Your code should be detailed enough that it is clear how SQL injections can be made. Explain your code in English. Give an example of a client side request that would exploit the vulnerability in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

b) Describe two different vulnerabilities that can result from allowing users to upload files, and explain how these vulnerabilities can be mitigated.

Question 5: Static analysis (7 points)

The following function computes the nth power of 2. Here, int denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows).

```
1  int exp(int n){
2    if(n < 0)
3      return -1;
4    int rstl = 1;
5    int i = n;
6    while (i != 0){
7      rslt = 2 * rslt;
8      i = i - 1;
9    }
10   assert(i == 0);
11   assert(rslt == 2^n);
12   return rslt;
13 }
```



We aim to check the assertions `(i == 0)` at line 10 and `(rslt ==` 2^n`)` at line 11. In the first part, we consider the following two approaches for checking a given assertion:

- Symbolic execution: builds a path formula obtained by violating the assertion after following a path through conditional statements (such as the one at line 2) and loops (such as the one at line 6) by choosing some outcome for the involved conditions.
- Abstract interpretation: here using the abstract values depicted in the lattice above. Intuitively, the abstract values are used to over-approximate, in an as precise manner as possible, the information of whether a variable is 0, positive, negative, or some combinations of these.

Questions:

1. Consider the assertion `(i == 0)` at line 10:
   (a) 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 10 and violating the assertion there (i.e. violating the `(i == 0)` assertion). (2 pt)
   (b) Can abstract interpretation, based on the sign abstract domain mentioned above, establish that the assertion `(i == 0)` is never violated? explain by annotating each line with the abstract element obtained at the end of such an analysis. (2pt)
2. Consider the assertion `(rslt ==` 2^n`)` at line 11:
   (a) Give the predicate P_7 defined as the weakest precondition of the predicate `Inv = ((rslt == 3*` 2^{n-i}`) && (0` $\leq$ `i) && (i` $\leq$ `n))` with respect to the assignment `i= i - 1` at line 8; then give P_6 defined as the weakest precondition of the predicate P_7 with respect to the assignment `rslt= 2 * rslt` at line 7? (2pt)
   (b) `Inv` is in fact an invariant of the loop (lines 6-9). Using this fact, can you argue why the assertion `(rslt ==` 2^n`)` at line 11 holds? (1pt)

Question 6: Security testing (7 points)

a) Briefly explain the difference between black-box and white-box testing techniques.

b) Consider cross-site scripting (XSS) vulnerabilities. Which of the two vulnerability types *Stored XSS* and *Reflected XSS* is generally easier to detect using a black-box web application fuzzer? Clearly motivate your answer.

c) A generation based fuzzer generally requires two components to work: A *grammar* and a set of *fuzzing heuristics*. Explain the purpose of both these components.

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

The code on the next page shows a function that prepends a prefix to each entry in a list of strings, before sending each string to a function `write_to_file`, the details of which are unimportant here. The function takes three parameters, a `prefix`, an array of strings (`str`), and the number of strings in the array (`n_strings`). It can be assumed that the number of strings in the array is always the same as the stated `n_strings`, but the contents of the prefix and the strings in `str`, as well as the number of strings in `str`, is user-controllable.

The function contains at least one serious bug that can lead to a potentially exploitable condition. Explain what the bug is, and how to fix it. Clearly explain what the consequence would be of triggering the bug.

```c
/* Writes 'size' bytes from 'data' to a predetermined file.
   Details not important here */
void write_to_file(const char* data, size_t size);

/* Takes an array of strings and a prefix, and prepends the prefix to
   each string before sending the resulting string to 'write_to_file'.
   Returns 1 on success, and 0 on failure. */
int append_prefix(const char* prefix, const char* str[], size_t n_strings)
{
   char buffer[256];
   char prefix_buffer[32];

   size_t prefix_len = strlen(prefix);

   strncpy(prefix_buffer, prefix, sizeof(prefix_buffer));
   prefix_buffer[sizeof(prefix_buffer)-1] = 0;

   // Replace all special (non-alphanumeric) letters in prefix with underscores
   for(size_t i = 0; i < prefix_len; i++) {
      if((prefix_buffer[i] < '0' || prefix_buffer[i] > '9') &&
         (prefix_buffer[i] < 'A' || prefix_buffer[i] > 'Z') &&
         (prefix_buffer[i] < 'a' || prefix_buffer[i] > 'z'))
      {
         prefix_buffer[i] = '_';
      }
   }

   for(size_t j = 0; j < n_strings; j++) {
      size_t str_len = strlen(str[j]);
      if(prefix_len > SIZE_MAX - str_len || prefix_len + str_len > SIZE_MAX - 1)
         return 0; // Integer overflow
      if(prefix_len + str_len + 1 > sizeof(buffer))
         return 0; // Too long strings

      strcpy(buffer, prefix_buffer);
      strcat(buffer, str[j]);
      write_to_file(buffer, prefix_len + str_len + 1);
   }

   return 1;
}
```