Information page for written examinations at Linköping University

Examination date	2017-04-21
Room (1)	TER2(19)
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	Ulf Kargén
Contact number during the exam time	013-285876
Visit to the examination room approximately	9:30, 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

2017-04-21

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) Consider the general software development lifecycle. In order to secure the lifecycle we can introduce security touch points. Draw the lifecycle and annotate where in the cycle you would use: *misuse cases*, *static analysis* and *penetration testing*.

b) During the release phase of SDL an incident response plan is made. In this course we have discussed four parts of this plan, describe these parts.

Question 2: Exploits and mitigations (5 points)

a) Briefly explain how a ROP-attack works, using English/Swedish *and* a figure.

b) Consider a Heartbleed-style vulnerability, which allows an attacker to read some data past the end of a buffer. This kind of vulnerability can be used to disclose sensitive information stored adjacent to the buffer. For each of the following two mitigations, explain whether it can mitigate this kind of attack, and why.

   i. ASLR

   ii. DEP

Question 3: Design patterns (5 points)

Explain the following two design patterns: *secure chain of responsibility* 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)

For the three (web)vulnerabilities: *SQL injection*, *command injection* and *cross-site scripting*, address the following (in the context of web security).

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

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

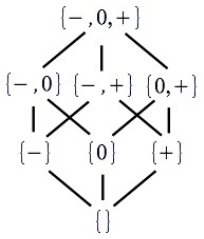
iii. Give an example of a request to the server-side code that would exploit the vulnerability (i.e. an attack).

iv. 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)

The type `int` denotes integers with an absolute value that can be arbitrarily large. Do not account for integer overflows.

```
1  int positive_euclidean_division(int a, int b){
2    if(a < 0)
3     return -1;
4    if(b < 0)
5     return -1;
6    int p = 0;
7    int r = a;
8    while (r >= b){
9      p = p + 1;
10     r = r - b;
11   }
12   assert(p >= 0);
13   assert(r >= 0);
14   return p;
15 }
```



We aim to check the assertions `(p >= 0)` at line 12 and `(r >= 0)` at line 13. 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 8) 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 `(p >= 0)` at line 12:
   (a) Give a **path formulas** that would correspond to taking the else outcome of the if statements (lines 2 and 4), entering the loop once (i.e., one iteration of the loop), exiting the loop to get to line 12 and violating the assertion there (i.e. violating the `(p >= 0)` assertion). (2 pt)
   (b) Can abstract interpretation, based on the sign abstract domain mentioned above, establish that the assertion `(p >= 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 `(r >= 0)` at line 13:
   (a) Let the predicate `Inv` be defined as `(r` $\geq$ `0)`. Give the predicate P_{10} defined as the weakest precondition of `Inv` with respect to the assignment at line 10 (i.e., `r= r - b`); then give P_9 defined as the weakest precondition of the predicate P_{10} with respect to the assignment at line 9 (i.e., `p = p + 1`). (1pt)
   (b) What does it mean for P_9 to be the weakest precondition of the predicate `Inv` with respect to the assignment sequence `r = r - b; p = p + 1`? (1pt)
   (c) Justify that the assertion at line 13 holds using your ansswer to question (2.b) and the relation between P_9 and the loop's condition at line 8. (1pt)

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) Give an example of a case where generation-based fuzzing would work well, but where mutation-based or whitebox fuzzing would not. Motivate clearly!

c) Explain *two* reasons why the scalability of concolic testing is limited for large and complex software.

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

Note: In the original version of the exam, this question had a typo that made the intended bug unexploitable, and also contained an unintended bug. This is the corrected version of the question.

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 an exploitable condition.

a) Explain what the bug is, and how to fix it. Clearly explain what the consequence would be of triggering the bug.

b) Provide an example input to the function that would trigger the bug. (You don't need to explain how to exploit the bug for e.g. code execution.)

```
/* 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));

   // Replace space with underscore in prefix
   for(size_t i = 0; i < sizeof(prefix_buffer); i++) {
      if(prefix_buffer[i] == ' ')
         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;
}
```