Information page for written examinations at Linköping University

Examination date	2018-08-29
Room (2)	TER1(11) TERF(1)
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 (available on phone) Alireza Mohammadinodooshan (visits exam room)
Contact number during the exam time	013-285876
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

2018-08-29

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ulf Kargén, 013-285876 (available on phone)

Alireza Mohammadinodooshan (visits exam room)

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) Is ASLR effective at mitigating a ROP attack? Explain why or why not. Be sure to mention enough details about how both ASLR and ROP works to clearly motivate your answer.

b) Give a high-level explanation of how stack cookies (a.k.a. stack canaries) work. Be sure to explain what kind of vulnerabilities stack cookies can mitigate, and how

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 motivation for the secure design pattern *clear sensitive information*. Give an example of a consequence of not using this pattern.

Question 4: Web security (6 points)

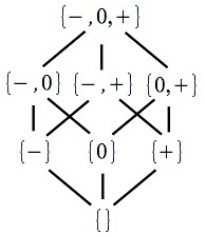
a) Give a complete example of how a CSRF attack could be carried out. Your answer should include an explanation of why the server-side code is vulnerable to CSRF, and how to mitigate the problem.

b) At least one of the (web) vulnerabilities discussed in this course can be used to stage a denial of service attack. Explain in detail how this can be achieved (using pseudo-code and English). Your response should clearly explain the vulnerability, the conditions for the attack to succeed and how the attack is performed.

Question 5: Static analysis (7 points)

The following function takes two integers `n` and `m` as input. It computes the remainder of the euclidean division of `n` by `m` in case both of them are strictly positive. It otherwise returns `-1`. For instance `remainder 10 2` gives 0 because 10=5*2+0 while `remainder 20 3` gives 2 because 20=3*6+2. Here, `int` denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows).

```
1  int remainder(int n, int m){
2    if(n <= 0 || m <= 0)
3       return -1;
4    int r = n;
5    int q = 0;
6    while (r >= m){
7      r = r - m;
8      q = q + 1;
9      assert(n = m * q + r);
10   }
11   assert(0 <= r);
12   return r;
13 }
```



We aim to check the assertions `(n = m * q + r)` at line 9 and `(0 <= r)` 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 condition (for example, choosing $(r < m)$ at line 6 in order to exit the loop and get to line 11).
- 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 `(0 <= r)` at line 11:
   (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 11 and violating the assertion there (i.e. violating the `(0 <= r)` assertion). (2 pt)
   (b) Can abstract interpretation, 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 and obtained at the end of such an analysis. (1pt)
2. Consider the assertion `(n = m * q + r)` at line 9:
   (a) Give P_8 defined as the weakest precondition of the predicate `(n = m * q + r)` with respect to the assignment `q= q + 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 - m` at line 7. (2pt)
   (b) Argue whether P_7 is 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. (2pt)

Question 6: Security testing (7 points)

a) Contrast *generation based* and *mutation based* fuzzing, and briefly describe their main strengths and weaknesses.

b) Briefly explain two general reasons (i.e. not related to specific vulnerability types) why automated fuzzing of web applications is often harder than fuzzing e.g. a desktop program written in C.

c) How does code-coverage feedback help a greybox fuzzer like AFL to find more bugs than a pure blackbox mutation based fuzzer?

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

The code on the next page shows part of a simple API for working with buffers containing Unicode text. The code has a serious bug, which could potentially be exploited by attackers.

a) Identify and name the vulnerability. Explain how an attacker could trigger the bug. You *don't* need to explain how to exploit the bug.

b) Explain how to fix the bug.

```
/*** External library dependencies ***/

/* Checks if pointed-to data is a valid Unicode character.
   Returns 1 if a valid character, 0 otherwise.
   Details of how this function works is unimportant. */
int isValidUnicode(const char* data);



/*** UnicodeBuf API definitions ***/

#define BUF_SIZE 1000000

/* Data structure to represent a buffer with Unicode data */
struct UnicodeBuf {
  char* buffer;         // Buffer to hold text data
  unsigned int n_used;  // Number of characters in buffer
};

/* Allocates and returns a new empty UnicodeBuf */
struct UnicodeBuf* UBCreate() {
  struct UnicodeBuf* b = malloc(sizeof(struct UnicodeBuf));
  b->buffer = malloc(BUF_SIZE);
  b->n_used = 0;
  return b;
}

/* Destroys an existing UnicodeBuf */
void UBDestroy(struct UnicodeBuf* b) {
  free(b->buffer);
  free(b);
}

/* Appends Unicode string (2 bytes per character) to UnicodeBuf.
   'data' can always be assumed to hold 2 * 'n_chars' bytes.
   Checks that each copied character is valid Unicode.
   Returns 0 on success, or non-zero to indicate an error. */
int UBAppend(struct UnicodeBuf* b, char* data, unsigned int n_chars) {

  if(b->n_used * 2 > BUF_SIZE - (n_chars * 2))
    return 1; // Too much data

  unsigned int i;
  for(i = 0; i < n_chars; i++) {
    if(isValidUnicode(data + 2 * i)) {
      b->buffer[2 * b->n_used]     = data[2 * i];
      b->buffer[2 * b->n_used + 1] = data[2 * i + 1];
    } else
      return 2;

    b->n_used += 1;
   }

  return 0;
}
```