Information page for written examinations at Linköping University

Examination date	2015-08-24
Room (1)	TER2
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

2015-08-24

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)

During the design phase of SDL, the goal is to describe how to securely implement all functionality provided by a given feature or function. In this course we discussed two specific tasks that should be completed as part of this phase. Name and briefly describe these two tasks (the descriptions shall be no longer than 30 words each).

Question 2: Exploits and mitigations (5 points)

For each of the two exploit techniques below, give a high-level explanation of how it works. Also explain which exploit mitigation it was specifically designed to overcome, and why it is able to do so.

a) Heap spraying

b) Return-oriented programming (ROP)

Question 3: Design patterns (5 points)

Explain the following two design patterns: *secure chain of responsibility* 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 the two (web)vulnerabilities: *cross-site request forgery* and *file inclusion*, address the following (in the context of web security). Use pseudo-code and/or English when appropriate.

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

b) Give an example of a flaw that causes this vulnerability.

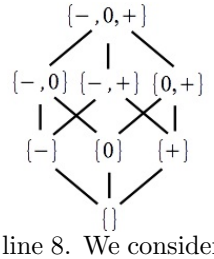
c) Give an example of a request to the server-side code that would exploit the vulnerability (i.e. an attack).

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, where int denotes integers with an absolute value that can be arbitrarily large (i.e., do not consider possibilities of integer overflows).

```
1 int foo(int x){
2   int y=1;
3   while (x < 100){
4     x = x + y;
5     y = y + 1;
6   }
7   y = x + y;
8   assert(y != 0);
9   ...
```



We aim to check the assertion `(y != 0)` at line 8. We consider the following two different methods:

- symbolic execution by checking the path formula obtained when unrolling the loop k times and violating the assertion
- abstract interpretation 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:

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 path formula that corresponds to violating the assertion after one iteration of the loop. Is the formula satisfiable? (1pt)

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

Question 6: Security testing (7 points)

a) Describe the main components in a typical fuzzing framework. Use **one or two sentences** per component. Overly long answers may lead to a reduction of points!

b) Imagine that you are tasked with creating a fuzzer for a JavaScript interpreter. Which fuzzing strategy (mutation or generation) would be most appropriate? Clearly motivate your answer!

c) What is the path-explosion problem in concolic testing?

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

The code on the next page demonstrates a simplified version of a security bug found in a well-known web browser last year. The vulnerable piece of code takes an array of C strings and concatenates them into one string, with each substring separated by a comma. (For example, if the input array is `{"aaa", "bbb", "ccc"}`, the output string would be `"aaa,bbb,ccc"`.)

a) Identify the vulnerability, and explain what the input should look like in order to trigger the bug. You **don't** need to explain how to exploit the vulnerability for e.g. arbitrary code execution.

b) Explain how to fix the bug.

Hint: The compiler used to compile the code uses the following sizes for built-in data types:

Data type	**Number of bits**	**Maximum size**
size_t	64	SIZE_MAX
long	64	LONG_MAX
unsigned long	64	ULONG_MAX
int	32	INT_MAX
unsigned int	32	UINT_MAX
short	16	SHRT_MAX
unsigned short	16	USHRT_MAX
char	8	SCHAR_MAX
unsigned char	8	UCHAR_MAX

```
// Joins all strings in the array 'strings' separated by commas and returns
// a pointer to a malloc-allocated buffer with the result. In case of error,
// NULL is returned.
// 'strings' is guaranteed to always hold exactly 'n_strings' strings,
// and all string pointers are guaranteed to be valid (e.g. not NULL).
// However, both the number of strings and string contents are user
// controllable.
char* join_strings(char* strings[], unsigned int n_strings)
{
   size_t total_strings_size = 0;
   size_t total_separators_size = (n_strings-1);
   size_t total_size = 0;
   unsigned int i;

   for(i = 0; i < n_strings; i++) {
      size_t size = strlen(strings[i]);
      if(total_strings_size > SIZE_MAX - size)
         return NULL; // Error
      total_strings_size = total_strings_size + size;
   }

   if(total_strings_size > SIZE_MAX - total_separators_size)
      return NULL; // Error
   total_size = total_strings_size + total_separators_size;

   return do_join(strings, n_strings, total_size);
}

char* do_join(char* strings[],
              unsigned int n_strings,
              unsigned int total_size)
{
   char* buffer;
   unsigned int i;

   // Allocate buffer, add 1 for NULL-terminator
   if(total_size > UINT_MAX - 1)
      return NULL; // Error
   buffer = malloc(total_size + 1);
   buffer[0] = 0; // Contains empty string initially

   // Concatenate strings and commas into 'buffer'
   for(i = 0; i < n_strings; i++) {
      strcat(buffer, strings[i]);
      if(i != n_strings-1)
         strcat(buffer, ",");
   }

   return buffer;
}
```