LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Nahid Shahmehri

Written exam

TDDC90 Software Security

2016-04-01

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)

Name and explain Phase 1 and Phase 3 of Microsoft's SDL (not counting the pre-SDL phase). Your explanations should be brief (maximum 60 words for each phase), but should account for the phases' activities.

Question 2: Exploits and mitigations (5 points)

a) 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.

b) Many practical implementations of stack cookies changes the order in which local variables are saved on the stack. Explain how the order of variables is changed, as well as the motivation for doing this reordering.

Question 3: Design patterns (5 points)

Explain the following two design patterns: *secure chain of responsibility* and *clear sensitive information*. 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)

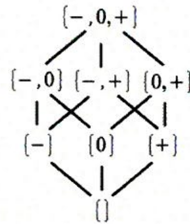
a) Explain how a slow HTTP-POST attack works. What is an attacker attempting to achieve by executing such an attack? How can the risk of such an attack be mitigated?

b) 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 (or Swedish). 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.

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., no integer overflows).

```
1  int foo(int n){
2    if(n < 0)
3       return 0;
4    int dec = n;
5    int inc = 0;
6    while (dec > 0){
7      inc = inc + 1;
8      dec = dec - 1;
9      assert((dec + inc) == n);
10   }
11   assert(dec == 0);
12   ...
```



We aim to check the assertions `((dec + inc) == n)` at line 9 and `(dec == 0)` at line 11. Given an assertion, we consider the following two approaches:

- 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 (`(n < 0)` at line 2 and `(dec > 0)` at line 6).
- 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 `(dec == 0)` at line 11:
   (a) Give a path formula that would correspond to taking the else outcome of the if statement (line 2), taking the loop once (i.e., one iteration), and violating the assertion at line 11. (1 pt)
   (b) Suppose `n` is positive, how many such paths are there (as a function of `n`)? (1 pt)
   (c) Suppose we want to check the assertion statically. Can symbolic execution exclude violation of the assertion at line 9 irrespective of the input? (1pt)
   (d) 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 obtained at the end of such an analysis. (1pt)
2. Consider the assertion `((dec + inc) == n)` at line 9:
   (a) Give `P`, the weakest precondition of the predicate `((dec + inc) == n)` with respect to the assignment `dec = dec - 1` at line 8? (1pt)
   (b) Give `Q`, the weakest precondition of the predicate `P` with respect to the assignment `inc = inc + 1` at line 7? (1pt)
   (c) Can abstract interpretation, based on the sign abstract domain mentioned above, establish the assertion is never violated? explain by using the annotations you used in question "1.(d)" above. (1pt)

Question 6: Security testing (7 points)

For each of the three following cases, explain and motivate which fuzzing technique out of *mutation-based fuzzing*, *generation-based fuzzing* and *whitebox fuzzing* that is ***least*** suitable. Also suggest one method (out of the preceding three) that you think would work better, and explain why.

a) Testing a closed-source spreadsheet editor with an undocumented file format.

b) Testing a server program implementing a stateful networking protocol.

c) Testing a file archiving tool that uses cryptographic hashes and digital signatures to verify the integrity of archives.

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

The code on the next page shows the beginning of a function that receives and processes a video stream from a potentially untrusted source over a network (e.g. the internet). The specific nature of the processing of video streams is not important here. The function contains at least one serious security bug.

a) Identify and name the vulnerability. Clearly explain how an attacker could trigger the bug.

b) Can the bug allow arbitrary code execution? If yes, briefly but clearly outline what an arbitrary code execution exploit for the bug would look like. If no, explain what the consequences of a successful exploit could be.

c) Explain how to fix the bug.

```
#define BUF_SIZE 2000000

// Represents a video stream. Details unimportant here.
struct Stream;

enum {
   QUALITY_LOW  = 1,
   QUALITY_MED  = 2,
   QUALITY_HIGH = 3
};

// Receive maximum 'size' bytes from stream 's' into memory pointed to
// by 'dst'. Returns the number of bytes actually read from stream. (Can
// be lower than 'size' if more data than what was available was
// requested.)
size_t receive(const struct Stream* s, size_t size, void* dst);

// Receives a video stream and processes it.
// Returns -1 in case of error, 0 otherwise.
int handleStream(const struct Stream* s)
{
   char buffer[BUF_SIZE];

   int quality;
   int n_seconds;
   int n_received;
   int data_rate;

   // Read header fields from stream:

   // First quality setting ...
   n_received = receive(s, sizeof(quality), &quality);
   if(n_received != sizeof(quality)) {
      printf("Transmission error!\n");
      return -1;
   }

   // ... and then number of seconds in stream
   n_received = receive(s, sizeof(n_seconds), &n_seconds);
   if(n_received != sizeof(n_seconds)) {
      printf("Transmission error!\n");
      return -1;
   }

   switch(quality) {
      case QUALITY_LOW:
         data_rate = 8192;
         break;
      case QUALITY_MED:
         data_rate = 16384;
         break;
      case QUALITY_HIGH:
         data_rate = 32768;
         break;
      default:
         printf("Unknown quality setting!\n");
         return -1;
   }

   if(n_seconds > BUF_SIZE / data_rate) {
      printf("Too much data!\n");
      return -1;
   }

   // Recieve stream data into 'buffer'
   n_received = receive(s, n_seconds*data_rate, buffer);
   if(n_received != n_seconds*data_rate) {
      printf("Transmission error!\n");
      return -1;
   }

   // Continue processing data...
```