LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Ulf Kargén

Written exam

TDDC90 Software Security

2023-03-17

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ulf Kargén, 013-285876

Instructions and grading

There are 7 questions on the exam. Your grade will depend on the total points you score. The maximum number of points is 38. The following grading scale is preliminary and might be adjusted during grading. You may answer in Swedish or English.

Grade	3	4	5
Points required	19	27	32

Question 1: Secure software development (4 points)

For each of the two modelling techniques *misuse cases* and *attack trees*, briefly (using a few sentences each) explain what the technique is used for. Also state in which part of the software development lifecycle it is most appropriate to use the technique, and briefly explain why.

Question 2: Exploits and mitigations (5 points)

a) Could ROP be used to exploit a buffer overflow on the *heap*? If no, explain why it is impossible. If yes, explain how.

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 or not it can mitigate this kind of attack, and why.

   i. ASLR

   ii. DEP

Question 3: Design patterns (4 points)

a) Explain the design pattern *secure factory*, including an explanation of why and when the pattern should be used.

b) Both of the patterns *privilege separation* and *defer to kernel* are specialized versions of a more general design pattern. Give the name of this general pattern, and explain the intent and motivation of it in 2-3 sentences.

Question 4: Web security (6 points)

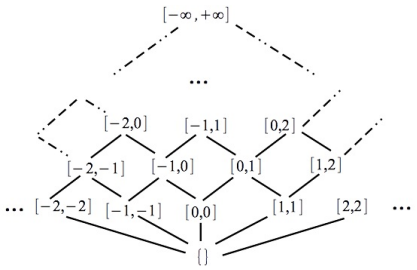
a) Consider a web app using Java as the server-side language. The web app needs to implement serialization of a complex data type (for example, a Java class), so that objects of the type can be received in web requests (for example, as part of an API). Clearly explain why it is inappropriate from a security point of view to implement this functionality with Java's own built-in serialization. Also, briefly explain how the functionality could be implemented in a secure way.

b) It has sometimes been incorrectly stated that disallowing GET-requests, and instead only allowing POST, is a way to mitigate *cross-site request forgery* (CSRF). Explain the rationale for this misconception. Also, give a counterexample of how CSRF-attacks can still be carried out.

Question 5: Static analysis (7 points)

Assume `int` denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows). Recall `%` is the modulo operator (`x % 2` returns 0 if `x` is even and 1 otherwise). Consider the following procedure.

```
1  int foo(int in){
2    int out = 0;
3    int i = 0;
4    while(i < in){
5      out = out + 2;
6      i = i + 1;
7    }
8    assert((out % 2) == 0);
9    assert(out >= 0);
10   return out;
11 }
```



We aim to check the assertions at lines 8 and 9 in procedure `foo`. Questions:

1. Symbolic execution:

   (a) Give, without checking its satisfiability, the SSA formula for the path condition that corresponds to a sequence of instructions starting at line 1 of procedure `foo`, and performing one iteration of the loop before violating the assertion at line 8. (1 pt)

   (b) Arbitrary many such path conditions can be generated (one for each number of iterations). Which ones are satisfiable? Explain. (1 pt).

2. Abstract interpretation: Annotate, after convergence of a most precise analysis, **the start of each line** in `foo` with an abstract element associated to **each one of the defined variables**. Use, for each variable, the interval domain, i.e., the abstract elements depicted in the lattice to the right of `foo`. (2pt).

3. The following part has 3 questions. The three answers result in a minimum of 0 pt and a maximum of 3 pts. Each wrong answer counts negative. E.g., one correct answer (1 x 1 pt), one wrong (1 x -1 pt) and not answering one (1 x 0 pt) result in a score of 0 pt out of the 3 possible points. Giving two wrong answers (2 x -1 pt) and one correct (1 x 1 pt) gives 0 points.

   Here, all variables are integers; `out = 0` stands for assignment and `(out == 2 * in)` stands for the predicate that evaluates to true iff `out` is twice `in`. State, without justification, whether each of the following Hoare-triples is valid or not.

   (a) $\{0 \leq in\}$ `out = 0; i = 0; while(i < in){out = 2 + out; i = i + 1;}` $\{out == 2 * in\}$

   (b) $\{0 > in\}$ `out = 0; i = 0; while(i < in){out = 2 + out; i = i + 1; }` $\{out == 0\}$

   (c) $\{true\}$ `out = 0; i = 0; while(i < in){out = 2 + out; i = i + 1;` $\{out \geq in\}$

Question 6: Security testing (6 points)

a) Which of *cross-site request forgery* (CSRF) and *SQL-injection* would be easier to detect (in the general case) using an automated web application fuzzer? Clearly explain your reasoning.

b) What is the path explosion problem in concolic testing?

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

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

The small C++ function shown on the next page reads text from a file, one line at a time, and concatenates all lines into one string before proceeding to process the concatenated text. (The nature of that processing is not relevant here.) The code contains at least one serious vulnerability, which could potentially be exploited to allow arbitrary code execution.

a) Identify the vulnerability and state the vulnerability type.

b) Explain what the input file should look like in order to trigger the bug. (You *don't* need to explain in detail how to exploit the vulnerability.)

c) Explain, in words *and* code/pseudocode, how to fix the bug.

You can assume that the comments in the code correctly describe the behaviour of library functions, etc. You don't need any additional knowledge about the library functions used than what is given in the comments.

```
void read_and_process(istream& in_file)
{
   const int MAX_DATA = INT_MAX;
   const int BUFSIZE = 1000;

   // Allocate a buffer on the heap with MAX_DATA bytes.
   char* concatenated = new char[MAX_DATA];
   // 'concatenated' should initially contain an empty string.
   concatenated[0] = 0;
   int total_read = 0;

   char buffer[BUFSIZE];

   // 'getline' reads one line of text (until a line delimiter is reached)
   // from 'in_file' into 'buffer'.
   // A maximum of 'BUFSIZE' bytes is written to 'buffer', including the
   // null terminator.
   // If end-of-file is reached, or the current line contains 'BUFSIZE'
   // characters or more, the call to 'getline' will evaluate to false,
   // and the loop will be terminated.
   while(in_file.getline(buffer, BUFSIZE))
   {
      int len = strlen(buffer) + 1;
      if(total_read + len > MAX_DATA)
      {
         printf("Error: Too much data");
         exit(1); // Quit program
      }

      // Append string in 'buffer' to existing string in 'concatenated',
      // starting from the position of the old null terminator in
      // 'concatenated'.
      strcat(concatenated, buffer);
      total_read = strlen(concatenated);
   }
   // Complete string read. Process it...
   process(concatenated);
}
```