Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2015-01-14
Sal	TER4/TERE
Tid	8-12
Kurskod	TDDC90
Provkod	TEN1
Kursnamn/benämning	Software Security
Institution	IDA
Antal uppgifter som ingår i tentamen	7
Antal sidor på tentamen (inkl. försättsbladet)	5
Jour/Kursansvarig	Ulf Kargén
Telefon under skrivtid	013-285876
Besöker salen ca kl.	9:00, 11:00
Kursadministratör (namn + tfnnr + mailadress)	Madeleine Häger Dahlqvist, 013-282360, madeleine.hager.dahlqvist@liu.se
Tillåtna hjälpmedel	Dictionary (printed, NOT electronic)

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

Written exam

TDDC90 Software Security

2015-01-14

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) In Security Development Lifecycle (SDL), how are bug bars and quality gates different?

Question 2: Exploits and mitigations (5 points)

Consider two different exploits for a stack-based buffer overflow vulnerability:

a) Buffer overflow leading to return pointer overwrite, using a NOP-sled and shellcode on the stack.

b) Buffer overflow leading to return pointer overwrite, using a ROP chain.

For each of the following mitigations, explain whether it would be effective at preventing the exploit from working, and why. You may need to make assumptions about the system and the particular implementation of a mitigation. Clearly state all such assumptions, and motivate your answers! Simple yes/no answers will give no points.

- Address space layout randomization (ASLR)
- Data execution prevention (DEP/W^X)
- Stack cookies

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)

The developers of a website are considering adding functionality that would allow users to upload files to the server, that later will be downloaded and used by other users. There are several vulnerabilities that could be introduced by allowing this. Please explain five of these. Your answer should include an explanation of the vulnerabilities, the possible consequences, and how they can be mitigated.

Question 5: Static analysis (7 points)

a) Splint is a static analyser. Is it sound, complete, or neither? Explain in one or two sentences.

b) Can a static analyser that is sound for some properties have false alarms about those properties? Explain in one or two sentences.

c) Explain in a sentence or two the problem with having false alarms.

d) Can symbolic executions have false negatives? If not, explain in one or two sentences, otherwise give a small example.

Question 6: Security testing (7 points)

a) Assume that you have been tasked with performing fuzz-testing of the two programs described below. For each program, explain if mutation-based fuzzing or generation-based fuzzing would be most appropriate.

   A. An FTP server. FTP is a well-known text-based protocol for transferring files, standardized in RFC 959. The FTP protocol has a complex state machine with many different message types for setting up a connection, requesting file transfers, etc.

   B. An image viewer and editor. The program uses a proprietary binary format for pictures, which is not officially documented by the vendor.

b) Explain why a simple depth-first search strategy for concolic testing may not find many bugs. Clearly explain your reasoning, possibly with a figure!

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

The small C++ function shown below 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 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.)

b) Propose, in words or 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 size_t MAX_DATA = 1000000;
   const size_t BUFSIZE = 1000;

   char concatenated[MAX_DATA];
   // 'concatenated' should initially contain an empty string.
   concatenated[0] = 0;
   size_t total_read = 0;
   // Allocate a buffer on the heap with BUFSIZE bytes.
   char* buffer = new char[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 more than
   // 'BUFSIZE' characters, the call to 'getline' will evaluate to false,
   // and the loop will be terminated.
   while(in_file.getline(buffer, BUFSIZE))
   {
      size_t len = strlen(buffer) + 1;
      // SIZE_MAX is the largest number that can be represented by 'size_t'
      if(total_read + len > MAX_DATA ||
         total_read > SIZE_MAX - len)
      {
         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'.
      // A maximum of 'MAX_DATA' characters will be copied from 'buffer'
      // into 'concatenated'.
      strncat(concatenated, buffer, MAX_DATA);
      concatenated[MAX_DATA-1] = 0;
   }
   // Complete string read. Process it...
   process(concatenated);
}
```