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

Distance exam

TDDC90 Software Security

2022-03-18

Teacher on duty

Ulf Kargén, ulf.kargen@liu.se, 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 36. 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

Answers should be submitted through Lisam as a single PDF or ASCII text (.txt) document before the end of the exam time. If you don't have access to a good software for making technical drawings that you are already familiar with, it is recommended to draw figures by hand and scan/photograph them. Figures could either be integrated into the PDF, or submitted as separate files in JPEG or PNG format. If you chose the latter approach, file names should be of the format Figure1, Figure2, and so on. All attached figures should be clearly referred to in the text by their file name. To facilitate easier grading, it is preferred that you express formulas, program code, etc. as text, and not as handwritten figures.

You are allowed to use any aid, but **all kinds of collaboration with others is strictly forbidden**. Also, **copying any part of an answer from another source will be considered plagiarism**. The questions will be checked for plagiarism and sharing of answers between students using Urkund, and any suspected cheating will be reported to the university disciplinary board. **By taking the exam, you solemnly promise to abide by the above rules.**

In the event that you experience technical difficulties with Lisam that prevent you from submitting, it is allowable to submit answers via email. However, this should only be used as a last resort if Lisam for whatever reason would stop functioning.

The distance exam will not be anonymous due to the exceptional COVID-19 situation.

Ulf Kargén will be available to answer questions during the exam via email and phone.

Question 1: Secure software development (3 points)

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.)

Question 2: Exploits and mitigations (5 points)

For each of the two cases below, come up with a small code/pseudocode example of a vulnerability *and* describe an attack against this vulnerability:

a) An attack mitigated by CFI but not by Stack Cookies.

b) An attack mitigated by Stack Cookies but not by DEP.

For each of the two cases, explain why the attack is prevented by the first mitigation but not by the second one. Make sure that the example code is detailed enough so that it is clear how an attack can be carried out.

Question 3: Design patterns (4 points)

Both the *Secure Factory* and *Secure Chain of Responsibility* design patterns can be used to address the same fundamental problem of (1) hiding the details and implementation of an access control policy from the calling object and (2) allowing dynamically changing the access control policy during runtime.

Both patterns are based on the premise that the calling object requests an action to be performed by passing some credentials to an external interface, and that the details of how the action is performed is determined based on the credentials. However, one of the patterns might be preferrable to the other, depending on the nature of the access control policy in use. Explain how the two design patterns differ in this regard. Also, for each of the two patterns, give a small example of an access control scenario that is more suitable to implement with that pattern, compared to the other.

Question 4: Web security (5 points)

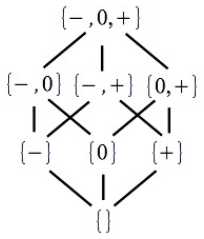
Server-Side Request Forgery (SSRF) is an attack where a public-facing web server is manipulated into performing a web request to an internal web server, which is not directly accessible to the attacker. This gives the attacker the ability to indirectly interact with the internal web server.

Name and briefly explain *three* different web vulnerabilities discussed in the course that can be used to stage an SSRF attack. Also give a high-level explanation of how each type of vulnerability can be exploited to perform SSRF.

Question 5: Static analysis (7 points)

Assume `int` denotes integers with an absolute value that can be arbitrarily large (i.e., no integer overflows). Consider the following procedure.

```
1   int foo(int x){
2     if(x < 0){
3       x = -x;
4     }
5     int q = 0;
6     int r = x;
7     while(r >= 10){
8       r = r - 10;
9       q = q + 1;
10      assert(r >= 0);
11    }
12    assert(x == 10*q + r);
13    return r;
14  }
```



We aim to check the assertions `(r >= 0)` and `(x == 10*q + r)` respectively at lines 10 and 12 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`, getting in the "then" branch of the if statement at line 2, and performing one iteration of the loop before violating the assertion at line 12. (1 pt)
   (b) Is this path condition satisfiable? Explain. (1 pt).
   (c) what is the main hinder to using symbolic execution to verify the assertion at line 12 holds for all inputs? Explain. (1 pt).

2. Abstract interpretation: Annotate, after convergence of the analysis, the end of each line in `foo` with the abstract element associated to each one of the defined variables. (1pt)

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. The following examples demonstrate the notation used:

   - `r=x` stands for assignment.
   - `res != 1` stands for testing inequality.
   - The sequence `q=0; r=x; while(r >= 10){r=r-10; q=q+1;}` represents sequential composition of two assignments followed by a while statement.

   State, without justification, whether each of the following Hoare-triples is true or false.

   (a) $\{x > 0\}$ `q=0; r=x; while(r >= 10){r=r-10; q=q+1;}` $\{x == 10 * q + r\}$
   (b) $\{true\}$ `q=0; r=x; while(r != 10){r=r-10; q=q+1;}` $\{r == 0\}$
   (c) $\{x < 0\}$ `q=0; r=x; while(r >= 10){r=r-10; q=q+1;}` $\{r < q\}$

Question 6: Security testing (6 points)

a) For each of the four fuzzer types *blackbox mutational*, *blackbox generational*, *greybox* and *whitebox*, explain whether or not that type of fuzzer would be suitable for testing a piece of code with lots of *magic constants*. Clearly motivate your reasoning, including an explanation of what magic constants are.

b) Consider a program with a Heartbleed-type vulnerability that can cause the program to read (not write) slightly beyond the end of a buffer, so that the contents of an adjacent buffer may be leaked. Even if the bug can be triggered by a fuzzer, it may still go unnoticed if the program is fuzzed using a traditional fuzzer setup. Explain why. Also, briefly describe a technique that can be used in combination with fuzzing to decrease the risk that such bugs are missed.

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 a serious security bug.

a) Identify and name the vulnerability.

b) Provided that you could host your own stream provider (and therefore have complete control over the streams), explain in detail how to craft a proof-of-concept exploit that redirects the flow of execution to an address of your choosing. (You only need to explain how to redirect execution, not how to actually get arbitrary code to execute.) State any assumptions you might make.

c) Explain how to fix the bug.

```c
#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...
```