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

Home exam

TDDC90 Software Security

2020-05-28

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 between students 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 home exam will not be anonymous due to the exceptional COVID-19 situation.

Ulf Kargén will be available to answer questions during the duration of 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)

Consider the simple function with a buffer overflow vulnerability below. An attacker can exploit the buffer overflow to achieve privilege escalation by overwriting the `full_priv` variable. For each of the following two mitigations, explain whether or not it would prevent the attack.

i. Stack cookies

ii. ASLR

Make sure to clearly motivate your answers and state any assumptions you make about how a mitigation is implemented. A simple yes/no answer without motivation gives no points.

```
void foo(char* user, int isAdmin)
{
   int full_priv = isAdmin;
   char buffer[16];
   strcpy(buffer, user);

   if(full_priv)
      // Do privileged stuff
   else
      printf("User %s is not admin \n", user);
}
```

Question 3: Design patterns (3 points)

A few years ago, it was discovered that a popular car model could be hacked remotely by exploiting a bug in the car's infotainment system, which was accessible through a 3G internet connection. The infotainment system was also connected to the car's internal CAN-bus, so that information about speed, fuel consumption, etc. could be displayed. By compromising the infotainment system, attackers could send arbitrary messages on the CAN-bus, which is used to control safety-critical systems such as breaks, steering, and transmission. Name a secure design pattern that could be used to mitigate the attack. Give a high-level description of how a system design adhering to this pattern would work, and why it mitigates the attack.

Question 4: Web security (6 points)

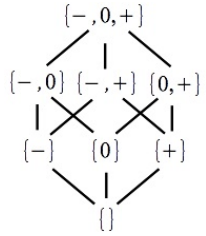
a) Using pseudo-code, write server-side code that contains a vulnerability that allows for SQL injections. Your code should be detailed enough that it is clear how SQL injections can be made. Explain your code. Give an example of a client-side request that would exploit the vulnerability in your code. Finally, suggest a modification of your code such that the vulnerability is removed. Explain why your mitigation strategy works

b) Reusing both the same username and the same password on several sites is a common security malpractice among users. This enables a particular kind of attack, discussed in the course. Name the attack, explain how it works, and state at least *two* methods (discussed in the course) to mitigate this kind of attack.

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, int y, int z){
2    if((x <= 0) || (y <= 0) || (z <= 0)){
3      return 0;
4    }
5    z = x - y;
6    if(x > 100){
7      x = 100;
8      if(y > 100){
9        y = 100;
10       if(x != y){
11         return x/z;
12       }else{
13         assert(z == 0);
14         return -1;
15       }
16     }
17   }
18 }
```



We aim to check the assertion `(z == 0)` at line 13. Questions:

1. Symbolic execution: Give a path formulas that would correspond to violating the assertion at line 13. (2 pt)

2. Abstract interpretation: can an abstract interpretation analysis based on the sign abstract domain mentioned above establish that the assertion is never violated? explain by annotating each line with the abstract element associated to each variable and obtained at the end of such an analysis (i.e., after the analysis converges). (1pt)

3. The following part has 4 questions. The four answers result in a minimum of 0pt and a maximum of 4pts. For clarity, we use "x := y" to mean that the value of y is assigned to x (i.e., assignment) and "x = y" to mean the boolean expression that checks whether the values of x and y are equal (i.e., test). Each wrong answer counts negative. E.g., two correct answers (2 * 1pt), one wrong (1 * -1pt) and not answering one (1 * 0pt) results in a score of 1pt out of the 4 possible points. Giving three wrong answers (3 * -1pt) and one correct (1 * 1pt) gives 0 points. State whether each of the following statements is true or false.

   (a) $\{true\}$ "x := y" $\{x \geq (y-1)\}$

   (b) $\{(x = 1) \wedge (y = 0)\}$ "x := y" $\{x \leq 1\}$

   (c) $\{(x \geq 1)\}$ "if (y = 10){x := 0}" $\{x \leq 0\}$

   (d) $\{(x \geq 1)\}$ "while(true){x := 1}" $\{x \leq 0\}$

Question 6: Security testing (6 points)

Consider an input format that uses a SHA256 checksum to verify the integrity of the input. Which of the fuzzing techniques discussed in the course (mutational blackbox, generational blackbock, whitebox, greybox) would be suitable for fuzzing a parser of this format, and which would not be suitable? Clearly motivate your answer for each of the four fuzzing techniques.

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

The code on the next page shows part of an API for working with lines of text in a simple text editor. The code contains a serious memory-corruption vulnerability. It can be assumed that an attacker can control the content of an edited text document, and that he/she will be able to request insertions and deletions of arbitrary size at arbitrary positions in a line of text.

a) Identify the bug in the code.

b) Explain how an attacker could trigger the bug.

c) What is the scope of the memory corruption, in other words, exactly what does the vulnerability allow the attacker to do?

d) Explain how to fix the bug.

You can assume that all comments describing the functionality of C library functions is truthful and correct.

```c
#define LINE_MAX 4096

// Data structure representing a line
struct Line {
   char line[LINE_MAX]; // char-array for holding contents
   size_t length;       // length of string _excluding_ 0-terminator
};

struct Line* new_line() {
   struct Line* l = malloc(sizeof(struct Line));
   l->length = 0;
   return l;
}

void destroy_line(struct Line* l) {
   free(l);
}

/* Inserts 'content' into line at 'insert_pos'. (First position is 0).
   Returns 1 on success and 0 on error.
   Example: line="abcdef", content="123", insert_pos=2
            result: "ab123cdef" */
int insert_in_line(struct Line* l, const char* content, size_t insert_pos) {
   if(insert_pos > l->length)
      return 0; /* invalid position */

   size_t content_sz = strlen(content);

   if(l->length > SIZE_MAX - content_sz || l->length + content_sz > SIZE_MAX - 1)
      return 0; /* integer overlfow */
   else if(l->length + content_sz + 1 > LINE_MAX)
      return 0; /* content won't fit */

   /* Copy trailing part (content to the right of inserted part).
      strdup first determines size of string by scanning until it finds
      0-terminator, and allocates just enough space for string+terminator
      using malloc. String is then copied into allocated memory. */
   char* temp = strdup(l->line + insert_pos);

   // Copy 'content' into line at 'insert_pos'
   strcpy(l->line + insert_pos, content);

   /* Copy back trailing part after new content
      (strcat scans destination string until it finds a 0-terminator,
      and copies in source string after that point) */
   strcat(l->line, temp);

   l->length += content_sz;

   free(temp);

   return 1;
}

/* Removes 'remove_sz' characters left of 'remove_pos' from line
   (like hitting backspace 'remove_sz' times).
   Returns 1 on success and 0 on error.
   Example: line="abcdef", remove_sz=3, remove_pos=4
            result: "aef" */
int remove_from_line(struct Line* l, size_t remove_sz, size_t remove_pos) {
   if(remove_pos > l->length)
      return 0; /* invalid position */

   // Copy remaining part to the right of 'remove_pos'
   char* temp = strdup(l->line + remove_pos);

   // Copy remaining part over removed part
   strcpy(l->line + remove_pos - remove_sz, temp);

   // Update length
   l->length -= remove_sz;

   free(temp);

   return 1;
}
```