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

Written exam

TDDC90 Software Security

2022-08-24

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)

a) Map each of the following two activities to the most appropriate phase of SDL, and explain why it is most appropriate to perform the activity during that phase: *fuzz testing* and *static analysis*.

b) In this course we mentioned three additional activities that can be requested by security advisors for projects using SDL. Name and briefly describe (using a sentence or two) *two* of these activities.

Question 2: Exploits and mitigations (5 points)

a) Provide a small code/pseudocode example of an *integer overflow* bug that could allow a *buffer overflow* attack (which would not otherwise be possible without the integer overflow). Explain how an attacker could perform a buffer overflow attack by triggering the bug. Also show how the same program could be modified to prevent integer overflows.

b) Most implementations of stack cookies also reorder the position of local variables on the stack in a specific way. Explain in what way variables are reordered, and the motivation for doing this.

Question 3: Design patterns (4 points)

a) One of the design patterns mentioned in the course literature can reduce the risk of introducing use-after-free (UAF) bugs. Name this design pattern and explain the main idea of it, and why it reduces the risk of having UAF bugs in your code.

b) Explain the motivation for the secure design pattern *secure logger*. Give an example of a consequence of not using this pattern.

Question 4: Web security (6 points)

a) Explain why using the HttpOnly flag when creating cookies in your web app can limit the consequences of an XSS bug in your app.

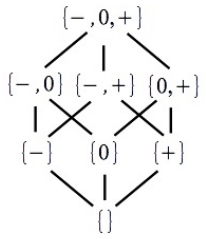
b) Explain what an *XML External Entities* (XXE) vulnerability is, and how it can be exploited by attackers. What is the underlying cause of XXE vulnerabilities, and how can this type of vulnerability be prevented?

c) Give an example of a concrete attack (i.e., something an attacker could achieve) that could be performed by exploiting an XXE bug.

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



We aim to check the assertions `(i >= 0)` and `(r == p*q)` 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) How many paths should symbolic execution check in order to verify the assertion at line 12 always holds? 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; `r=r+p` stands for assignment. The sequence `i=0; r=0; while(i < q){r=r+p; i=i+1;}` represents sequential composition of two assignments followed by a while statement. State, without justification, whether each of the following Hoare-triples is valid or not.

   (a) $\{q \geq 0\}$ `i=0; r=0; while(i < q){r=r+p; i=i+1;}` $\{r == i * p\}$

   (b) $\{true\}$ `i=0; r=0; while(i < q){r=r+p; i=i+1;}` $\{r == i * p\}$

   (c) $\{q < 0\}$ `i=0; r=0; while(i < q){r=r+p; i=i+1;}` $\{r < q\}$

Question 6: Security testing (6 points)

a) Which of the two vulnerability types *Cross-Site Request Forgery* (CSRF) and *SQL injection* would be easier to detect with an automated web application fuzzer? Clearly motivate your answer.

b) Greybox fuzzing is often exponentially faster at finding bugs than a simple mutational black-box fuzzer. Explain why, using a small code/pseudocode example.

c) When using a greybox fuzzer to test a program whose input format uses *checksums*, it is typically necessary to patch out (i.e., disable) the checksum validation code in the program. Briefly explain why.

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

The code on the next page shows the beginning of a program that performs some privileged operation, the nature of which is not important here. The program makes sure that the user is allowed to perform the privileged operation by first asking for a password and user name, and checking these against a user database. (Exactly how the database works is also not important here.) The program has a very serious implementation error, which could allow attackers to gain unauthorized access.

a) Identify the bug and name the vulnerability type. Explain how an attacker can exploit the vulnerability to gain unauthorized access. You may need to make some assumptions about how specific system components work. Clearly state these!

b) Explain how to fix the bug.

```c
// Read input string from user and store in 'dst'. A maximum
// of 'max_size' bytes will be written to 'dst', including the
// NULL terminator. ‘dst’ is guaranteed to always be NULL-terminated.
// Returns 0 if successful, or -1 if the input does not fit in 'dst'.
int get_input(char* dst, size_t max_size);

// Returns the user ID for the given user name, or -1 if the
// user name is unknown.
int get_user_id(const char* username);

// Fetches the password for given user ID from the database and stores
// it in 'dst' as a NULL-terminated string. Passwords cannot be longer
// than 20 characters. Therefore, a maximum of 21 bytes can be written
// to 'dst'. Returns 0 if successful, or -1 if the operation failed due
// to an invalid user ID.
int get_password(char* dst, int user_id);

int main(void) {
   const size_t BUF_SIZE = 256;
   int id = -1;
   int failed = 0;

   printf("Username: ");
   char* user = malloc(BUF_SIZE);
   if(get_input(user, BUF_SIZE) != 0) {
      // Too long input
      printf("Error: Too long username!\n");
      free(user);
      exit(1); // Quit program
   } else {
      id = get_user_id(user);
   }

   printf("Password: ");
   char* given_password = malloc(BUF_SIZE);
   if(get_input(given_password, BUF_SIZE) != 0) {
      // Too long input
      printf("Error: Too long password!\n");
      free(user);
      failed = 1;
   }

   char* real_password = malloc(BUF_SIZE);
   if(get_password(real_password, id) != 0) {
      failed = 1;
   } else if(strcmp(real_password, given_password) != 0) {
      // If real password is not identical to given password,
      // authentication fails (strcmp returns 0 if strings are equal).
      failed = 1;
   }

   if(failed == 0)
      printf("Authenticaton succeeded for user %s!\n", user);
   else {
      printf("Authenticaton failed for user %s!\n", user);
   }

   // Zero out password in memory
   memset(real_password, 0, BUF_SIZE);

   if(failed == 1) {
      exit(2); // Quit program if authentication failed
   }

   // Continue processing...
```