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

Home exam

TDDC90 Software Security

2020-08-26

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 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)

For each of the two modelling techniques *misuse cases* and *attack trees*, state in which part of the software development lifecycle it is most appropriate to use the technique. Briefly motivate your answers

Question 2: Exploits and mitigations (5 points)

a) Some older implementations of ASLR only randomized the position of the stack, and not other sections of memory. Would such an ASLR implementation still be effective against a ROP-exploit of a stack-based buffer overflow? Clearly motivate your answer.

b) Name one attack technique that is prevented by CFI but not by DEP. Explain why CFI is effective for stopping the attack and why DEP isn't.

Question 3: Design patterns (3 points)

The two similar design patterns *Privilege Separat*ion and *Defer to Kernel* are both special cases of the more general pattern Distrustful Decomposition. Briefly discuss in which cases it is more appropriate to use one or the other.

Question 4: Web security (6 points)

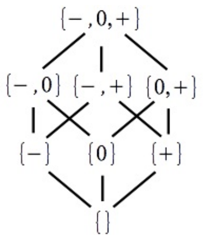
a) Explain why using the HttpOnly flag when creating cookies in your web app can limit the consequences of an XSS bug in your app.

b) Give a complete example of how a CSRF attack could be carried out. Your answer should include an explanation of why the server-side code is vulnerable to CSRF, and how to mitigate the problem.

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 void foo(int x, int y, int z){
 2   if((x <= 0) || (y <= 0)){
 3     return;
 4   }
 5   if(x > 100){
 6     if(y > 100){
 7       if(x + y == z){
 8         assert(z > 201);
 9       }else{
10         z = z - y;
11         assert(x != z);
12       }
13     }
14   }
15 }
```



We aim to check the assertions `(z > 201)` at line 8 and `(x != z)` at line 11. Questions:

1. Symbolic execution: Give a path formula that would correspond to violating the assertion at line 11. (1 pt)

2. Abstract interpretation: show that the sign abstract domain mentioned above cannot establish that the assertion at line 8 is never violated 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. Abstract interpretation: propose an abstract domain that would establish the assertion at line 8 and use the domain to annotate 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)

4. The following part has 4 questions. The four answers result in a minimum of 0 pt and a maximum of 4 pts. 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 expression that tests whether the values of x and y are equal. We use $\vee$ for disjunction in the predicates. Each wrong answer counts negative. E.g., two correct answers (2 * 1 pt), one wrong (1 * -1 pt) and not answering one (1 * 0 pt) results in a score of 1 pt out of the 4 possible points. Giving three wrong answers (3 * -1 pt) and one correct (1 * 1 pt) gives 0 points. State whether each of the following statements is true or false.

   (a) $\{(x = 5) \vee (y > 2)\}$ "x := y" $\{(x > 3) \vee (y = 3)\}$

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

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

   (d) $\{(x \geq 0)\}$ "while(x $\geq$ 0){x := x + 1}" $\{x < 0\}$

Question 6: Security testing (6 points)

a) 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.

b) Consider cross-site scripting (XSS) vulnerabilities. Which of the two vulnerability types Stored XSS and Reflected XSS is generally easier to detect using a black-box web application fuzzer? Clearly motivate your answer. (Make sure to include an explanation of the difference between Stored and Reflected XSS in your motivation.)

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

The code on the next page shows a function `login` for handling password checks as part of a login system. The function first queries for the user name, and then gives the user the option to either type in the password, cancel the login, or request that a password reset link be sent to a provided email address (given that the provided email address matches with the one associated with the username). The code contains a serious security bug. (We here ignore the other glaring security issue that passwords are stored in plain text.)

a) Identify the bug in the code.

b) Clearly explain with an example how an attacker could exploit the bug, and what the consequences of a successful exploit would be.

c) 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 USR_LEN 16
#define PWD_LEN 16
#define EMAIL_LEN 32

// Fetches password for given username and stores it into 'destination'
// Passwords are never longer than 15 characters.
// Returns 0 on success and 1 on error.
int get_pwd_from_user_db(const char* username, char* destination);

// Sends a password-reset link to given email, provided that the email is
// registered with the given username. Returns 0 on success and 1 on error.
int request_email(const char* username, const char* email);

// Performs credentials check. Returns 0 on successful login and 1 on
// failed login or error.
int login()
{
   char username[USR_LEN];
   char given_password[PWD_LEN];
   char email[USR_LEN];
   char stored_password[PWD_LEN];
   int choice;

   printf("User name: ");
   // Read string from standard input. Given format specifier "%Ns" fscanf will
   // read at most N characters into destination buffer ('username' here), or
   // until whitespace (e.g. newline) is encountered. A null terminator is
   // always appended after the last read character in the destination buffer.
   fscanf(stdin, "%15s", username);
   if(get_pwd_from_user_db(username, stored_password) != 0) {
      printf("Error accessing user database\n");
      return 1;
   }

   // Loop until explicitly terminated with return statement
   while(1) {
      printf("Choose action:\n");
      printf("   1: Proceed to type password\n");
      printf("   2: Cancel login\n");
      printf("   3: Request password reset email\n");
      // Read one integer from standard input into 'choice'
      fscanf(stdin, "%d", &choice);

      if(choice == 1) {
         printf("Password: ");
         fscanf(stdin, "%15s", given_password);
         // strcmp returns 0 if strings are equal
         if(strcmp(given_password, stored_password) == 0) {
            printf("Login successful!\n");
            return 0;
         } else {
            printf("Incorrect password!\n");
            return 1;
         }
      } else if(choice == 2) {
         return 1;
      } else if(choice == 3) {
         printf("Email address: ");
         fscanf(stdin, "%31s", email);
         if(request_email(username, email) != 0) {
            printf("Error sending email\n");
            return 1;
         } else {
            printf("A password reset link has been sent!\n");
         }
      } else {
         printf("Invalid choice, please try again.\n");
      }
   }
}
```