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

Written exam

TDDC90 Software Security

2023-08-23

Solution hints

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

Important: Note that the purpose of the solution hints is to provide guidance when using the old exam for self-study. For full score on an exam, more detailed/elaborate answers than the one-sentence hints given here would typically be required.

Question 1: Secure software development (4 points)

Many software vendors today use a mechanism where parts of the functionality of an application (e.g., a document viewer) is disabled for untrusted files, and the user needs to click a button to activate all functionality.

a) What is the purpose of this mechanism?
Reduce the amount of code exposed to potential exploits

b) Which important security principle is this an example of? Explain the general principle using a sentence or two.
Attack surface reduction…

c) In which phase of the SDL is application of this principle mandated?
Phase 2 (Design)

Question 2: Exploits and mitigations (5 points)

Using pseudocode, provide an example of a program with a *heap-based buffer overflow* bug that could be exploited for *arbitrary code execution*. Clearly explain why the code is vulnerable, and how an arbitrary code execution attack would be carried out. (You don't need to provide a detailed exploit.)

The example program should be *complete*, i.e., the pseudocode should roughly mirror a C/C++ source file that implements all functionality of the program. (You don't need to provide pseudocode for common library functions, just the core functionality of the program.)

Solution would need to demonstrate an overflow into a heap-allocated object/struct that contains either a function pointer or virtual functions (i.e., has a VTable entry).

Question 3: Design patterns (4 points)

a) Explain the intent and motivation of the *Resource Acquisition Is Initialization* (RAII) pattern. Name a type of vulnerability that is avoided by using this pattern.
Intent: Make sure that allocated resources are deallocated under all possible program execution paths. Motivation: Avoid confusion about which part of a program that is responsible for deallocating resources, and thereby avoid potential memory leak, use-after-free, double-free bugs. Use-after-free and double-free would be acceptable answers for security bugs prevented by the pattern.

b) The two similar design patterns *Privilege Separation* 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.
Privilege Separation: When high-priv and low-priv components can operate mostly independently, with occasional interaction.
Defer to Kernel: When low-priv component frequently need to request support from high-priv component.

Question 4: Web security (6 points)

a) Give *two* examples of web attack types discussed in the course that can lead to arbitrary code execution on a *web server*, and briefly explain how they work.
Command execution and insecure deserialization are the two most obvious examples.

b) Explain how the *HTTP Strict Transport Security* (HSTS) mitigation works. What kind of vulnerability does it mitigate?
HTTP header that instructs browser to only accept TLS-protected requests. Prevents malicious redirection to http://... URL instead of https://...

Question 6: Security testing (6 points)

a) 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.)
Reflected XSS is easier, since script is immediately reflected back in response. Triggering Stored XSS is a two-step process (get script into database, then trigger response containing script)

b) What is *Penetration testing*? State one benefit and one disadvantage of Penetration testing.
Manually trying to “hack” a system, in order to expose security bugs. Advantage: can detect problems that are impossible to find with automated methods. Con: expensive, results dependent on pentester’s skill level.

c) What are the two main black-box fuzzing techniques called? What is the principal difference between them? Also, state the main disadvantage of each technique
Mutation-based fuzzing: Randomly mutate valid seed inputs, con: poor coverage
Generation-based fuzzing: Generate semi-valid inputs from formal input specification, con: much manual effort often required to craft input specification.

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

The code on the next page shows part of a program that reads data from a simple data stream format. Each message in the stream consists of two 32-bit integers, followed by a variable-size data part. The first number specifies the type of data in the message, and the second number specifies the length of the data. The data part then follows directly.

The part of the program we are concerned with (`print_text_record`) parses a message, and prints its data if it is of text type (plain text or XML). We also give function declarations for a simple API for reading data from streams. It can be assumed that the API functions are correctly implemented, and function in accordance with the comments given above each function declaration. The program may be used to read streams from untrusted sources.

`print_text_record` contains at least one serious bug that can lead to a potentially exploitable condition.

a) Identify and explain the vulnerability in the code.
`size` can be negative, leading to buffer overflow when calling `DStream_read_data`.

b) Explain the consequences of a successful exploit. (I.e., what could an attacker achieve by triggering the bug?)
Stack-based buffer overflow (with complete control over the amount of overwritten memory) allows arbitrary code execution via return pointer overwrite.

c) Clearly explain how to modify the code to fix the bug.
For example, use `text_size` instead in `if(size >= 1024)`.

```c
/* Constants defining message data types. */
enum DStream_Types {
   DS_PLAIN_TEXT  = 1,
   DS_XML_TEXT    = 2,
   DS_RGB_RASTER  = 3
};


/* Reads 4 bytes off the stream 'ds' and returns them as an int. */
int DStream_read_int32(struct DStream* ds);


/* Reads 'sz' bytes from stream 'ds' into buffer 'dest' */
void DStream_read_data(struct DStream* ds, char* dest, size_t sz);


/* Check if a previous operation on 'ds' has caused an error.
   Returns zero (false) in case of no error, and non-zero otherwise. */
int DStream_has_error(struct DStream* ds);


int print_text_record(struct DStream* ds)
{
   char buffer[1024];

   int type = DStream_read_int32(ds);
   int size = DStream_read_int32(ds);

   // Check if we failed to read header from stream
   if(DStream_has_error(ds))
      return -1;

   size_t text_size = size;

   if(text_size <= 0)
      return -1; // Invalid size

   if(type == DS_PLAIN_TEXT)
      printf("Type: TEXT\n");
   else if(type == DS_XML_TEXT)
      printf("Type: XML\n");
   else
      return 1; // Not a text-type record

   // Check that input fits in buffer
   if(size >= 1024)
      return -1; // Oversized record

   // Read into buffer
   DStream_read_data(ds, buffer, size);

   if(DStream_has_error(ds))
      return -1; // Error reading string off stream

   buffer[size] = 0; // Make sure string is terminated

   // Print string
   printf("%s\n", buffer);

   return 0; // Signal success
}
```