Speed Wels
5136905

Exam Software Security, 201600051

University of Twente

31 January 2018, 13:30-16:30

This exam consists of 9 questions of equal weight. It is a closed-book exam: **the use of any printed or online material is prohibited**. Short and to-the-point answers are highly preferred over long stories.

1. Spotting low-level vulnerabilities in C.

   (a) The following code-fragment in C has three `printf`-statements. Which of these is the most secure one? Which is the least secure? Why?

```
int main (int argc, char* argv[]) {
  char s1[] = "%s, %d\n";
  const char s2[] = "%s, %d\n";
  char* s3;
  s3=argv[1];

  external_function(s1,s2,s3);

  printf(s1,"exam",50);
  printf(s2,"software",50);
  printf(s3,"security",50);
}
```

   (b) The following C-fragment first copies 10 characters from the input to buffer, and then copies buffer to dest.
   Explain why this can still lead to a buffer overflow?

```
  char buffer[10], dest[10];
  strncpy(buffer,argv[1],10);
  strcpy(dest,buffer);
```

2. Low-level attacks and counter-measures.

   (a) What is the difference between a "code injection attack" and a "code reuse attack"? To which category does "return-to-libc" belong?

   (b) How do "stack canaries" (partially) protect against both of these attacks?

   (c) "Non-executable memory" protects only against one of these attacks. Which one? (explain your answer).

3. Memory- and type-safety in programming languages.

   (a) When is a programming language "memory-safe"?

   (b) When is a programming language "type-safe"?

   (c) Which of these two notions is stronger? Why?

   (d) Provide an example of violation of type-safety in C?

4. Rust as a secure programming language.

   (a) Why are variables immutable by default in Rust?

   (b) How does the ownership concept of Rust avoid (many) memory leaks?

   (c) Ownership also prevents programming bugs due to aliasing. How can ownership be transferred permanently or temporarily?

5. McGraw introduces security touchpoints all over the software life cycle.

   (a) Mention two of McGraw's touchpoints in the early design phase (when identifying requirements and use cases)?

   (b) Mention two touchpoints in the late design phase (when the code exists already)?

   (c) Explain the difference between security flaws and security bugs?

   (d) When does a flaw/bug become a security vulnerability?

   (e) How do you identify good test cases for security testing?

6. The AFL Fuzzer generates and improves a set of test cases by "genetic programming".

   (a) Mention two different mutations that AFL applies to the current generation of test cases, in order to generate new test cases?

   (b) What is the (fitness) criterion for AFL to decide which test cases should be selected?

   (c) Fuzzing with AFL reports "crashes" and "hangs" in the program under test. Mention two methods that can be combined with AFL to discover more security problems?

7. Symbolic execution generates constraints on symbolic input variables while executing a program.

   (a) Consider the following (pseudo)-code. Under which constraint can the assertion be reached?

```
x := input();
y := input();
if (x>y)
   if (x+y>3)
      return OK;
   else
      assert ERROR;
else
   return OK;
```

   (b) How are the generated constraints solved in practice, to get concrete test input data?

   (c) Explain the "path problem" for symbolic execution?

   (d) Another issue for symbolic execution is that library code is unknown or very large. Which technique extends symbolic execution to handle library calls?

8. Input validation and web programming.

(a) Mention three countermeasures against command/filename injection attacks?

(b) What is the difference between CSRF (cross-site request forgery) and XSS (cross-site scripting) injection?

(c) What does SOP (same-origin-policy) enforce, and why can it be bypassed by XSS?

9. (a) Consider the following code fragment in PHP. Why is this insecure? (refer to the erroneous line number)

```
1:  $dir = $_GET['option']
2:    if ($dir = 1) {
3:     include("1/function.php")
4:  } else {
5:  if ($dir = 2){
6:     include("2/function.php")
7:  } else {
8:     include("error/$dir/unknownvalue.php")
9:  }}
```

(b) Traditional scripting languages use string concatenation with place holders to construct SQL queries. What support do modern programming languages (such as Wyvern) offer to construct queries in a more secure manner?

(c) What is the best practice for positioning input validation code within a large program?

(d) What is the essence of a second-order SQL injection?