*NB These old exam questions do not include questions on new reading and/or lecture material.*

# Mock Exam Software Security

Be concise and to the point: most questions can be answered in one or two sentences. Good luck!

1. **Memory unsafety and countermeasures**

   a) There are two flaws related to memory management in the code below.

   ```
   char* src = malloc(9);
   char* domain = "www.ru.nl";
   strncpy(src, domain, 9); // copies 9 bytes from domain to src
   ```

   Say what these flaws are, and how you would fix them. (Using `malloc` rather than `calloc` could be considered a 3rd flaw, but that this is not one of the flaws we are looking for.)

   b) Do you think that PREfast could catch these two mistakes? Motivate your answers.

   c) Can stack canaries offer some protection against return-to-libc attacks? Motivate your answer. (NB the question is not if stack canaries provide perfect protection against all possible return-to-libc attacks, just if they can offer some protection in some cases.)

   d) Could PREfast spot the buffer overflow in the code below, if the code is not annotated using SAL? Or could it only if the code *is* annotated using SAL?

   Motivate your answers. If you think annotations would be needed, say which ones. (No need to get the PREfast syntax anywhere near right; just say where annotation(s) are needed and what these should express.)

   ```
   void main() {
       int elements[100];
       initialise(elements, 100);
     }

   void initialise(int* buf, int len){
      int i;
      for(i = 0; i <= len; i++) buf[i] = 0;
   }
   ```

2. Describe two notions of memory safety, explaining what guarantees they offer and what the difference is between them.

   (If you can only think of one notion of memory safety, describe that one.)

3. **Vulnerabilities and countermeasures**

   a) What is a TOCTOU attack? As part of you answer, also give an example of such an attack.

   b) What is the difference between a white-listing and a black-listing approach in input validation? Which of the two is better, and why?

4. **Java sandboxing**

a) In granting permissions, Java's sandboxing mechanism considers the permissions not just of the top stack frame (which is the stack frame of the method currently executing), but potentially of all frames on the stack.

Why would a design where permissions are granted just on the basis of the permissions of the top stack frame not be right, in that it could not provide the right security? (Giving an example may be the easiest way to to explain this.)

b) Suppose we have a trusted Java class `T` and an untrusted class `U` with a policy giving only `T` the right to write to a file `secret.xls`:

```
grant codebase T.java
   { permission java.io.FilePermission "secret.xls","write"; };

class U{
   T t;
   void try0() { ... // does something that uses privilege P}
   void try1() { t.m1(); }
   void try2() { t.m2(); }
   void try4() { t.m2(); t.m1(); }
   public void main(String[] args) { ... }
}

class T{
  final static Permission P = new FilePermission("secret.xls", "write");
  void m1() { ... // does something that uses privilege P}
  void m2() { enablePrivilege(P);
              ....// does something that uses privilege P }
}
```

For each of the methods of `T`, say whether it will result in a security exception or not. Here assume that these methods are called from the `main` method in `T`, (so that `main`'s activation record is the bottom frame on the call stack).

Briefly motivate your answers.

5. **Information flow** We consider a type system for confidentiality as in Chapter 5 of the lecture notes. We distinguish two levels, $L$ for public data and $H$ for confidential data. Program variables are either $L$ or $H$, meaning they are for storing public data (L) or confidential data (H).

Type judgements are of the form $e : T$ and $s : \mathsf{ok}\, T$ where $T \in \{L, H\}$, where

- if $e : T$ then expression $e$ only contains information of level $T$ or lower, and
- if $s : \mathsf{ok}\, T$ then statement $s$ does not leak confidential information to public variables and all assignments in $s$ are to variables of level $T$ or higher.

So, if $e : L$ then also $e : H$, and if $s : \mathsf{ok}\, H$ then also $s : \mathsf{ok}\, L$.

a) Provide the typing rule (or rules) for an if-then-else statement, of the form below but with the question marks filled in:

$$\frac{e :? \quad s_1 : \mathsf{ok}\,? \quad s_2 : \mathsf{ok}\,?}{\texttt{if (}e\texttt{) then } s_1 \texttt{ else } s_2 : \mathsf{ok}\,?}$$

if we ignore both (non)termination and timing as a channel for leaking information.

b) Explain what implicit information flows are for if-then-else statements, e.g. by giving an example of how such an implicit flow can happen, and explain how your rule(s) rules disallow implicit flows.

c) Now give the rule(s) for if-then-else if we are worried about (non)termination as a channel for leaking information. Motivate your answer, by explaining why the changes w.r.t. your answer for part a) are needed and how these avoid information leaks by (non)termination.

6. **Non-interference**

   Consider a program $P$ operating on just two variables, a high variable $h$ with confidential information and a low variable $l$ with public information that may be observed by anyone.

   For program states $s_i$, which assign values to these two variables, we define $s_1 =_L s_2$ to mean that $s_1$ and $s_2$ give the same value to $l$ and $s_1 =_H s_2$ to mean that $s_1$ and $s_2$ give the same value to $h$.

   (a) Define what it means for program $P$ to be non-interferent (i.e. not to leak information from $h$ to $l$), where we ignore non-termination and timing as covert channels.

   (b) Now assume that the program might throw exceptions. Define what it means for program $P$ to be non-interferent where we still ignore non-termination and timing as covert channels, but where we do consider exceptions as covert channel that leak information.

7. The paper 'Collaborative Verification of information flow for a High-Assurance App Store' by Michael Ernst et al. describes the SPARTA approach to certifying security properties of Android apps.

   Explain in a few sentences how this approach works.

   In your description, make it clear

   - which steps are manual and which are tool-supported;
   - which parts of the process can be done by the app-developer (who wants his app verified) and which parts have to be done by the owner of the app-store (who wants to guarantee that apps in his store adhere to a sensible policies);
   - which annotations have to be written by the app-developer, and which annotations have to be provided by the app-store.

8. Suppose that for some protocol you know the format that messages have and the order in which these messages have to be sent. If you have to test an application that implements this protocol for security flaws, how would you do this?

   Mention at least two different forms of security testing that you could use for this, and explain the different kinds of security flaws that these forms of testing can reveal.

9. Explain why the LangSec ideas are particularly relevant for buffer overflow problems – both in explaining root causes and in suggesting countermeasures. (So in your answer, discuss how root causes identified by LangSec also cause many buffer overflows, and how can good practises advocated by LangSec can help to prevent buffer overflows.)

**The remaining questions are only for the 6EC version of the course!**

10. **JML and ESC/Java2**

Which JML annotations (`invariant`, `requires`, or `ensures` clauses) would be needed to check with ESC/Java2 that the code below never throws a runtime exception? Hint: there are two types of runtime exception we'd like to avoid here.

```
/**  Represents a set of digits, i.e. a subset of {0,1,2,..,9} */
public class SetOfDigits {

 /** The array 'contains' records which digits are contained in the set.
   * Eg. the array
   *    {true,false,true,false,false,false,false,false,false,true}
   * represents the set
   *    {0,         2,                                       9}
   */
 private boolean[] contains = new boolean[10];

 public void add     (int i) {contains(i) = true;}
 public void remove  (int i) {contains(i) = false;}
 public boolean contains(int i) {return (contains(i));}
}
```