# Security in Software Applications

Matteo Piermartini, Aurora Polifemo, Edoardo Ottavianelli

## 1. SHORT QUESTIONS

1. **Describe one notion of memory safety explaining what guarantees it offers.**
   With safety we mean a system protected against accidental failures. So a programming language is memory-safe if it guarantees that
   1. programs can never access unallocated or deallocated memory, hence no segmentation faults at runtime and so OS access control for memory is not necessary (if there are no bugs in the execution engine).
   2. (maybe) programs can never access uninitialized memory, so there is no need to zero-out memory before deallocating to avoid information leaks (if there...)
2. **What is the difference between a normal SQL injection attack and a blind SQL injection attack?**
   In a normal SQL injection we input a payload breaking the SQL query logic (mix code and data) and the web application sends us the result of the query (it could be data, error string, error code...), thus we can exploit easily the webapp retrieving all the data. Instead in a blind SQL injection we input the payload, but the application doesn't send us back the result of the query. We can infer it using conditions in the query having understandable results: a boolean value, a cookie being set, time of response...
3. **What is the difference between a black-listing and a white-listing approach to input validation?**
   Both of these are solutions for input validation. Basically we write a list of elements and the program will check this list for allowed/disallowed inputs. In white-list the elements written in the list are the allowed ones, and anything that is not in the list will be disallowed. In a black-listing approach is the complete opposite, the elements in the list are the disallowed ones and anything not in the list will be allowed. There isn't a 'better' approach, so they are both useful, it just depends on the situation, but keep in mind it's easier to miss blocked inputs in a black-listing approach (they could lead to vulnerabilities).
4. **Name <u>two</u> techniques for code obfuscation and <u>two</u> measures of code complexity (relative to obfuscation).**
   Two techniques for code obfuscation are:
   1. *<u>Layout obfuscation</u>*: changes or removes useful information, without affecting real instructions;
   2. *<u>Data Obfuscation</u>*: changes the way data is implemented.

3. *Control-flow obfuscation*: Affects the control-flow within the code. Intervenes in the way the different instructions are executed.

The measures of code complexity are:

a. Program length: Number of operators and operands
b. Data flow complexity: Number of inter-block variable references
c. Cyclomatic complexity: Number of predicates in a function
d. Nesting complexity: Number of nesting level of conditionals in a program
e. Data structure complexity: Complexity of the static data structures in the program like variables, vectors, records
f. 00 Metrics: Level of inheritance, coupling, number of methods triggered by another method, non-cohesiveness.

**5. Describe three Principles of Secure Design**

Choose three :)

a. Least privilege – Each user and program should operate using the fewest privileges possible. Limits the damage from an accident, error, or attack. Reduces the number of potential interactions among privileged programs. Unintentional, unwanted, or improper uses of privilege less likely. Extend to the internals of a program: only smallest portion of program which needs those privileges should have them
b. Economy of mechanism/Simplicity – Protection system's design should be simple and small as possible – “techniques such as line-by-line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.“ – Aka “KISS” principle (``keep it simple, stupid'').
c. Open design – The protection mechanism must not depend on attacker ignorance. Instead, the mechanism should be public. Makes it possible for users to convince themselves the system is adequate. Not realistic to maintain secrecy for a distributed system.
d. Complete mediation (“non-bypassable”) – Every access attempt must be checked; position the mechanism so it cannot be subverted. For example, in a client- server model, generally the server must do all access checking because users can build or modify their own clients.
e. Fail-safe defaults (e.g., permission-based approach) – The default should be denial of service, and the protection scheme should then identify conditions under which access is permitted – More generally, installation should be secure by default.
f. Separation of privilege – Ideally, access to objects should depend on more than one condition, so that defeating one protection system won't enable complete access.
g. Least common mechanism – Minimize the amount and use of shared mechanisms (e.g. use of the /tmp or /var/tmp directories) – Shared

objects provide potentially dangerous channels for information flow and unintended interactions

h. Psychological acceptability/Easy to use – The human interface must be designed for ease of use so users will routinely and automatically use the protection mechanisms correctly. Mistakes will be reduced if the security mechanisms closely match the user's mental image of his or her protection goals.

6. **What is the difference between a heap buffer overflow and a stack buffer overflow?**
   Heap BO = buffer overflow in buffer located dynamically in the heap
   Stack BO = buffer overflow in buffer located in the stack to hold variables
   HBO is more difficult to exploit than SBO. An integer overflow can lead to a (heap) buffer overflow.
7. **What is a TOCTOU attack? Give a simple example as part of your answer**
   TOCTOU (Time Of Check, Time Of Use) is a source of security problems based on non-atomic check and use. Some precondition required for an action is invalidated between the time it is checked and the time the action is performed.
8. **Describe one of the rules for Secure Java Programming**
   The rules are:
   - Never return (a reference to) a mutable object (incl. arrays) to untrusted code
   - Never store a reference to a mutable object (incl. arrays) obtained from untrusted code
   - Don't use inner classes
   - Make classes non-cloneable
   - Make classes non-deserializable
   - Make classes non-serializable
   - Limit access to classes, methods, and fields, make them private, unless... there is a need for them to be more visible
   - Make classes, methods, fields final, unless there is a need for them to be subclassed/overridden/modified.
   - Avoid reflection, the ability of a program to inspect & modify itself. Reflection allows access to fields that are normally not visible.
9. **Given `int {o1:r1,r2;o2:r2,r3} x` in JFlow, show 2 possible declassifications of x**
   Declassification allows more access, either by adding one reader or by removing one policy
   - int {o1 : r1, r2 ; o2 : r1, r2, r3 } x
   - int {o1 : r1, r2 ; } x
10. **Can stack canaries offer some protection against return-to-libc attacks? Motivate your answer. Note: 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.**

Yes, it offers some protection if the attacker is not able to "guess" the value of the canary, because to perform these kinds of attacks the stack needs to be overwritten, so if we place a canary (random value) before the return address, that value would be overwritten too. So we could check if it has been modified before executing the return address.

11. **Why do "secure Java programming" guidelines warn against providing support for de/serialization?**
    a. **Give an example of a class for which de/serialization might give problems.**
    b. **Explain which capabilities an attacker must have to carry an attack abusing deserialization (i.e. describe the attack model).**

Serialization may leak confidential information, allowing an attacker to view the internal state of objects (private portions included). An attacker can create a sequence of bytes that happens to deserialize to an instance of that class with arbitrary values (i.e. deserialization is a kind of public constructor).
(a) A class with an array object is serialised. An attacker can modify its length to Integers.MAX_VALUE and, when deserialized, exhaust the VM memory resulting in a DoS attack.
(b) The attacker must have access to a remote service which accepts untrusted data for deserialization, or access to the classpath of an application which includes serializable classes

12. **Explain why the language-based security ideas are particularly relevant for buffer overflow problems, both in explaining root causes and in suggesting countermeasures.**

Mechanisms to provide safety include:

- compile time checks (e.g., type checking)
- run time checks (for array bounds, null pointers, runtime type checks, ...)
- garbage collector for automated memory management (no need to free() dynamic memory)
- execution engine: in JVM, bytecode verifier to type-check code, runtime checks, garbage collector invoked periodically

Due (mainly) to the first two points, it's difficult to achieve a buffer overflow, where we should overwrite memory locations out of the bounds of an array.
We should use modern strongly-typed languages that can guarantee language-based security (like JAVA/.NET) and avoid weakly-typed languages; moreover buffer overflow security is enforced by a combination of static and runtime checks.

13. **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 have to 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. With this system, can information be whitewashed?**
    1. The developer manually adds annotations (@Nullable, @NonNull...) and checks for declassifications, while checker frameworks are used to automatically check these.
    2. The developer provides: app description, Information Flow Policy, Annotated Source Code and Declassification justifications. The app store: checks if information policy is acceptable, runs the type checker and checks the declassifications.
    3. The app-developer can write annotations regarding the flow of the data, focusing on the input and the output of each information (@Source and @Sink). Types of Sources: Camera, Location, SMS... and Types of Sinks: Display, Internet, FileSystem..etc. Information can be whitewashed by storing and then reading it from the filesystem, but this transitive flow has to be explicitly stated (e.g. Camera -> FileSystem; FileSystem -> Internet needs explicit policy Camera -> Internet).

14. **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.**
    We may do Fuzz testing: a fuzzer is an automated tool which provides invalid/random inputs to a program and can spot problems in handling these corner cases. Inline testing allows the verification of invariants within a program (more code we execute, more bugs we find), but with this approach some rules and assumptions are difficult to test.
    We could also use Final Testing: test cases run against the entire project looking for bugs added when all the sections of the application are chained together. Another way to test the application is using Integration Testing: test for discrepancies in assumptions and security models; ensure that all

the operations that should be denied are denied, verify that partial accesses can only access what they should.

## 2. VULNERABILITIES IN CODE

- **(10+5 points) Find a vulnerability in the code and suggest changes to remove it. Do you think that Flawfinder would have found it? Motivate your answer.**

```
char *stringcopy(char *str1, char *str2) {
        while (*str2) *str1++ = *str2++;
        return str2;
}
main(int argc, char **argv) {
        char *buffer = (char *)malloc(16 * sizeof(char));
        stringcopy(buffer, argv[1]);
        printf("%s\n", buffer);
}
```

The second row copies elements from the second buffer (str2, the source) in the first one (str1, the destination) till there are elements available, but the function does not check if the destination has enough space to allocate all the elements. We can exploit this by overwriting the buffer (buffer overflow) injecting a NOP sled, a shellcode and a return address pointing to the code.

To solve this problem we should use safe functions like strncpy, or at least check the length of the destination while iterating over the source buffer.

Flawfinder will not catch this vulnerability because it is a custom implementation and this tool checks for known functions (strcpy, strcat...).

- **Find the vulnerabilities in this code and suggest changes to remove them. Do you think that Flawfinder would have found them? Motivate your answer.**

```
#define LEN 200
void fill(int *buf, int len){
    int i;
    for(i = 0; i<=len; i++) buf[i]=1;
}

void work() {
    int elements[200];
    int j;
    for(j = 0; j <= LEN; j++) elements[j] = 0;
    fill(elements, LEN);
}

void work_more() {
    char* string = (char*) malloc(3);
    string[0] = 'r';
    string[1] = 'u';
```

```
    string[3] = 'n';
}
```

1. 4th line: we should check if len is greater than buf (buffer overflow)
2. for loop in work(): it should be j < LEN (buffer overflow)
3. Last line: string[3] is out of buffer bound (buffer overflow)
4. string is not terminated (‘\0’ missing)

Flawfinder is not able to spot these kind of error, since it simply matches for e.g. use of unsafe functions from standard library.

- **Find the vulnerability in the code and suggest changes to remove it. Do you think Flawfinder would have found it? motivate your answer**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char* str)
{
    char buffer[24];
    strcpy(buffer, str);
    return 1;
}
int main(int argc, char** argv)
{
    char str[517];
    FILE* badfile;
    badfile = fopen("badfile", "r");
    fread(str, sizeof (char), 517, badfile);
    bof(str);
    printf("returned properly\n");
    return 1;
}
```

1. fread doesn’t add the terminator, we should set 516 as input of fread and then add manually the terminator: str[516] = ‘\0’;
2. Since there isn’t a terminator in str, strcpy is vulnerable to buffer overflow, so we should use strncpy.
3. We should also check returned values for errors
   Flawfinder would be able to spot the second one, but not the others.
4. When calling bof(str), we need to consider the fact that the buffer in the bof function is long 24, instead str is 517, so we need to use strncpy and specify the length of the buffer. But in this case the src is longer than the dest, so the null terminator is not being added by the function, so we use 23 and add the terminator. strncpy(buffer, str, 23) and buffer[24] = ‘\0’ (or strncpy(buffer, str, 22) and buffer[23] = ‘\0’).

- **There are two flaws related to *memory management* in the code below. Both of these flaws might cause this code to trigger a segmentation fault, if we are lucky enough that the segmentation protection will detect them.**

```
char* src = malloc(9);
```

```
char* domain = "www.ru.nl";
strncpy(src, domain, 9);
```

**Say what these flaws are, and how you would fix them.**
First, the string domain is long 9 bytes plus 1 byte for its termination character '\0', so we have to allocate 10 bytes of memory for src (as to avoid a buffer overflow when copying the whole string). Second, strncpy with length 9 does not copy the termination character '\0', and accessing the string src can result in undefined behaviours. Note: strncpy() is not considered a safe syscall, it would be better to use snprintf(); or use it, but remember to leave the last byte for the null terminator.

## 3. JAVA STACK WALKING/INTROSPECTION

1. **Describe what the Java Stack Walking (aka Inspection) algorithm is and how it works in JVM. 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 does not provide the right security?**
   It is an algorithm for granting permissions based on stack inspection: every resource access or sensitive operation requires having an appropriate permission P.
   new thread inherits access-control context of parent thread:
   def demandPermission(Permission P):
   FOREACH call on stack:
   IF caller lacks P THEN THROW EXCEPTION
   IF caller has P disabled THEN THROW EXCEPTION
   IF caller has P enabled THEN RETURN
   check inherited access-control context
   Suppose we have two classes A and B, where the former has a method with permissions to access some kind of restricted resource, while the latter does not have these permissions. If we check only the first frame, then we cannot distinguish between class A invoking its safe method or class B invoking (in a previous frame) the safe method from class A.

2. **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 = new T();
        void try0() { // does something which uses privilege P };
        void try1() { t.m1(); };
        void try2() { t.m2(); };
```

```
                void try3() {
                        t.m2();
                        t.m1();
                };

        public void main(String[] args) {}
        }
        Class T {
                final static Permission P = new FilePermission("secret.xls",
"write");
                void m1() { // does something which uses privilege P };
                void m2() {
                        enablePrivilege(P);
                        // does something which uses privilege P
                };
        }
```

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

1. try0(): SecurityException, as class U does not have privilege P
2. try1(): SecurityException, as privilege P is not enabled for the caller U
3. try2(): execution continues, as privilege P has been enable
4. try3(): same as try2(), since privilege P has not been disabled between calling t.m2() and t.m1()

## 4. OPENJML

1. **Which JML annotations would be needed (with OpenJML) to enforce Persons are either male or female and can only marry people of the opposite sex.**

```
class Taxpayer {
       //@ invariant isFemale == True => isMale == False;
       //@ invariant isFemale == False => isMale == True;
       //@ invariant isMale == True => isFemale == False;
       //@ invariant isMale == False => isFemale == True;
      boolean isFemale;
      boolean isMale;
       int age;
      boolean isMarried;
      Taxpayer spouse;

      Taxpayer(boolean babyboy) {
            age = 0;
            isMarried = false;
            this.isMale = babyboy;
            this.isFemale = !babyboy;
```

```
            spouse = null;
      }

       //@ requires this.isMale == True => new_spouse.isFemale == True;
       //@ requires this.isFemale == True => new_spouse.isMale == True;
      void marry(Taxpayer new_spouse) {
            spouse = new_spouse;
            isMarried = true;
      }

      void divorce() {
            spouse.spouse = null;
            spouse = null;
            isMarried = false;
       }
}
```

2. **Add the appropriate JML annotations (ESC/Java2 compatible) to detect and remove the errors in the following code and to enforce the constraint.**

```
class Set {
    //@ invariant 0 <= n <= elements.length;
    int[] elements;
    int n;
    void delete(int item) {
        //@ loop_invariant 0 < n < elements.length;
        for (int i=0; i<=n; i++) {
            if (elements[i] == item) {
                n--;
                elements[i] = elements[n];
                return;
            }
        }
    }
    //@ requires 0 <= n < elements.length;
    boolean contains(int item) {
        for (int i=0; i<=n; i++) {
            if (elements[i] == item) {
                return true;
            }
            return false;
        }
    }
    //@ requires (2 * (\bigint) n) + 1 <= Integer.MAX_VALUE;
    //@ when n == elements.length;
    void add(int item) {
        if (n == elements.length) {
```

```
            // should be 2*n+1
            int[] newContents = new int[2*n];
            arraycopy(elements, 0, newContents, 0, n);
            elements = newContents;
        }
        elements[n]= item;
        n++;
    }
}
```

**3. 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 exceptions 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.
       *  E.g. The array {True,False,t,f,f,f,f,f,f,t}
       *  represents the set {0, 2, 9}.
       */
      private /*@ non_null @*/ boolean[] contains = new boolean[10];

      //@ requires 0 <= i < 10;
      public void add(int i) { contains[i] = true; }
      //@ requires 0 <= i < 10;
      public void remove(int i) { contains[i] = false; }
      //@ requires 0 <= i < 10;
      public boolean contains(int i) { return (contains(i)); }
}
```

## 5. INFORMATION FLOW

1. **(6+8 points) Consider a program P with only two variables, H (with confidential information) and L (with public information). s1=H s2 indicates that the two states s1 and s2 associate the same value for the variable high; similarly for s1=L s2.**
   a. **Define what it means for the program P to be non-interferent (ignoring non-termination and timing)**
   b. **Take into account non-termination. Define what it means for the program P to be non-interferent considering non-termination as a covert channel**

c. **Now assume that the program may throw exceptions. Define what it means for the program P to be non-interferent considering exceptions as covert channel**

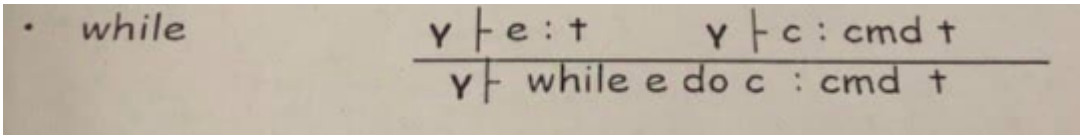
a. A program P does not leak information if, for all s1 =L s2: if executing P in s1 terminates and results in s1', and executing P in s2 terminates and results in s2', then s1' =L s2'.

b. A program P does not leak information if, for all s1 =L s2: if executing P in s1 terminates in s1', then executing P in s2 also terminates, and results in some s2' with s1' =L s2'.

c. (probabilistic ni?)

2. **In the type system approach to information-flow control, illustrate and justify**
**(1) the "while" rule**
**(2) the "if-then-else" rule.**
**Which is the main problem with these rules? How can they be fixed? Give an example.**
These rules state that:
(1) we can assign type 'cmd t' to the loop [ while (e) do c ] provided that c is of type 'cmd t' and e is of type 't' (prevents explicit flows)
(2) we can assign type 'cmd t' to the branching [ if (e) then c1 else c2 ] provided that c1 , c2 are of type 'cmd t' and e is of type 't' (prevents explicit flows) and can prevent explicit flows. This does not rule out non-termination and implicit flows as a covert channel, e.g. leak information about the confidential guard H:
(1) (non-termination) while (H) do {. . .} and if (H) then {//loop} else {//terminate}
(2) (implicit flows) while (H) {L = x} and if (H) then {L = x} else {L = y}
It can be fixed by setting t = L.

3. **(15 points) In the type system approach to information flow, <u>illustrate and justify</u> the rule**




**Which is the main problem with this rule? How can it be fixed?**
Look at the previous exercise: both first points.

4. **Provide the typing rule(s) for an "if-then-else" statement, of the form below but with the question marks filed 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.**

If we ignore the non-termination and timing as covert channels we should substitute t as type for e, while `cmd t` for s1 and s2 (and of course cmd t also at the end of the second statement). Instead if we do not ignore them, we should substitute L (low, public type) as type for e.

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

1. if (H=99) then {loop} else {terminate} (non-termination)
2. if (H=1234) then {...} else {...} (execution time)

For the rule that avoids these types of implicit flows see the answer above.

**6. 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.**

See the answer 4.