1. Secure software use of secret values is an important aspect of overall software security engineering. Please answer the following questions about the handling of keys by software.

(a) Discuss a security issue related to long-term key storage.

**Answer:** Long-term storage of keys, specially in disk, must take into account handling of back-ups and possible direct access to the underlying media. Therefore, keys’ long term storage should take place in encrypted format. For instance, an administrative password may be used to encrypt the keys. Even better, keys could be encrypted using an administrative key that is physically protected in physically secured storage (such as a smartcard). Therefore, software should be programmed to handle encrypted key storage, not raw keys.

(b) Discuss a security issue related to the handling of keys in memory. Do the same concerns apply to short-term values that should remain secret, such as session keys and/or nonces?

**Answer:** Every buffer that stores both long-term secret such as keys as well as short-term (but sensitive) values such nonces and session keys needs to be explicitly zeroed before the corresponding memory is de-allocated. This prevents the memory from being allocated to a different process, who proceeds to read from the newly allocated memory positions before initializing those memory addresses. An even better practice is to zero the buffer after its last use (which may come much earlier than de-allocation).

(c) Why is it difficult to hide a hard-coded password or key in binaries?

**Answer:** Many tools, such as de-assemblers and de-compilers, can go a long way towards making binary code more legible and help identify stored secrets.

2. Describe the principle of least-privilege.

**Answer:** Services and applications should be given the minimum set of privileges and access needed to accomplish their tasks.

3. What is chroot and how can it be used to implement the principle of least privilege?

**Answer:** chroot is a Unix command that changes the value of the file-system root directory for a particular application. This restricts an application of a limited view of the file system, and in particular prevents it from having access to any files outside the root specified by the chroot command.

Usage:
4. Many network and operating system services need to bind to a low-digit port number (≤ 1024). In Unix, an application must have root privileges to request a low-digit port number. How can this be reconciled with the principle of least privilege? Assume that the applications do not need root privilege for any other reasons during its operating uptime.

**Answer:** Applications can give up their root privileges immediately after acquiring the required port. So they initialize with root privileges and may go through privilege de-escalation by using the command "; ".

5. The popular SMTP agent **Sendmail** was configured to run with root privileges for the duration of its uptime, when indeed it only required root privileges to do the initial binding to a low-digit port. This is an example of a massive violation of the principle of least-privilege. This created many critical vulnerabilities in systems that ran the agent. For instance, buffer overflow attacks that enabled the attacker to run arbitrary shell commands in privileged mode and completely hijack the target machine.

6. A special Unix group is used to facilitate the application of the least-privilege principle. If a program is given effective group id equal to the nobody group, it is prevented from even having access to world-readable files in the system.

7. In making decisions about how to secure systems, it is vital to perform a detailed assessment of vulnerabilities and threats attending to the operation of the system. The use of information acquired through such risk analysis can inform important strategic decisions. In particular, effort can be wasted if the weakest link, or most-easily exploitable vulnerability, is not addressed first.

8. Security, as safety, requires redundancy. The use of layered security measures to guarantee fallback options when the first security measure fails, is called defense-in-depth.

9. Describe the principle of failing securely and give an example.

**Answer:** It is important that programmers and system administrators anticipate consequences of program failure. For instance, in some cases it is required to change access privileges to files where memory dumps are saved, for if a user can force a program to crash, important information, such as higher-privilege keys, may be contained in the dump.

10. Describe how failure of negotiating a protocol version can lead to vulnerabilities.

**Answer:** For reasons of interoperability with earlier protocol versions, server security updates often accept older (and often insecure or broken) versions of a protocol after an upgrade to a newer version. An infamous example of that is LM hashed-password authentication. An attacker can pretend to implement an obsolete client and force a server to engage in an unsafe session.

11. Describe a design choice that eliminates the need for backward-interoperability that can be a source of vulnerabilities in protocol version negotiation.

**Answer:** It is a better design to anticipate the need for security upgrades and to implement clients that can automatically (and securely) download patches when they detect newer server protocol versions.

12. **Black-box testing** is a security-testing technique that does not require detailed specifications (except for things such as a user manual). Teams tend to discover random faults that can be exploited for security. It is both a robustness and security testing process.
13. **Red teaming** consists in assembling groups of computer experts and challenging them to penetrate a system, with knowledge restricted to the type of information available to a typical attacker.

14. **What is the Common Criteria?**

   **Answer:** It is a standard for verification of implemented software design and development processes. It does not guarantee that a particular piece of software is secure, but that every stage of the software development process is documented to conform with specific criteria. It is an International Standard, with National or regional variations called profiles.

15. **What is a buffer overflow attack and how can it be implemented?**

   **Answer:** A buffer-overflow attack consists of an attacker providing large inputs to an application that do not fit in the reserved memory spaces and spills over adjacent memory regions. In order for a buffer overflow to be successful, an attacker must accomplish two things. (1) Re-write memory sections with code instructions for the attack, and (2) cause a change in program flow control to execution of the instructions in the re-written portion of the memory.

16. **What is the stack and why can it be exploited via buffer overflow attacks?**

   **Answer:** The stack contains activation frames for functions and procedures. An important part of a stack frame is the return pointer, which stores the memory position of the next executable instruction upon completion of execution of the function or routine in the current stack frame. A buffer overflow attack can overwrite the return pointer with the address of the beginning of the malicious code provided by the adversary.

17. **When may a piece of code be vulnerable to a buffer overflow attack?**

   **Answer:** When it fails to do array bounds checking on all inputs provided by another party.

18. **Why type-safe languages such as Java are less vulnerable to buffer overflow attacks? Are they completely secure against such attacks?**

   **Answer:** Type-safe languages do array-bounds checking for all variable assignments, either through static checks or dynamic ones. However, the native implementations of type-safe languages runtime environments, and their dependency on native implementations of system libraries mean that they may still be indirectly vulnerable to buffer overflow attacks.

19. **Why assessment tools such as RATS can only find previously known buffer overflow threats?**

   **Answer:** General-case array bounds checking is an undecidable problem. Since no general-case checking algorithm is available, instead assessment tools use a database of known unsafe coding patterns.

20. **What are array bounds-checking compilers? What are the drawbacks of using such tools?**

   **Answer:** These compilers introduce dynamic checks, refusing to make assignments that overflow buffers. It prevents against all overflows, but it can cause significant performance decrease in code that makes many memory accesses and memory writes.
21. **What is static analysis? What are the drawbacks of its use?**

**Answer:** Static analysis can enforce the use of coding instructions that are verifiably secure, i.e., are known not to cause array-bound checking errors. This restricts the type of instructions that software developers can use, reducing productivity and sometimes efficiency of generated code. It also cannot protect legacy code that has already been compiled (such as system libraries, drivers, etc).

22. **What is a canary value? How can it protect against stack smashing attacks?**

**Answer:** A canary value is a value that is checked before the function is allowed to return using the address in the return pointer. The canary is placed along the path of the buffer overflow so that it will be overwritten before the return address is. (At least if the overflow takes place in the direction of string growth, most easily exploitable overflow case.) The code needs to be re-compiled using a modified compiler to introduce the proper checks in the generated assembly code.

23. **Why cannot a canary value protect against all buffer overflow attacks?**

**Answer:** It only protects code re-compiled with canary protection, not linked libraries. The canary value can be guessed if the method for generating its value is not strong, so the attacker can make sure it is overwritten with its correct value (with some probability).

