

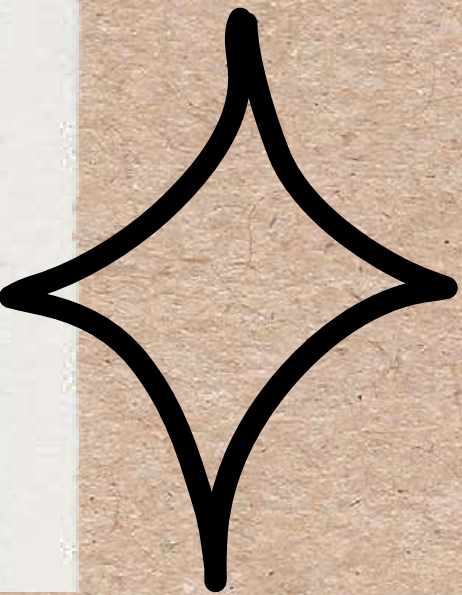


SOFTWARE ENGINEERING. ASSIGNMENT #06

20B030299

Rymbayeva Anelya

7IT, 3 course



13.8 Explain why it is important when writing secure systems to validate all user inputs to check that these have the expected format.



A common attack on a system involves providing the system with unexpected inputs that cause it to behave in an unanticipated way. These inputs may simply cause a system crash, resulting in a loss of service, or the inputs could be made up of malicious code that is executed by the system. Buffer overflow vulnerabilities, first demonstrated in the Internet worm and commonly used by attackers, may be triggered using long input strings. So-called SQL poisoning, where a malicious user inputs an SQL fragment that is interpreted by a server, is another fairly common attack. You can avoid many of these problems if you specify the format and structure of the system inputs that are expected. This specification should be based on your knowledge of the expected system inputs. For example, if a surname is to be input, you might specify that all characters must be alphabetic with no numbers or punctuation (apart from a hyphen) allowed. You might also limit the length of the name. For example, no one has a family name with more than 40 characters, and no addresses are more than 100 characters long. If a numeric value is expected, no alphabetic characters should be allowed. This information is then used in input checks when the system is implemented.

14.2 What are the types of threats that have to be considered in resilience planning? Provide examples of the controls that organizations should put in place to counter those threats

Three types of threats have to be considered in resilience planning:

1. Threats to the confidentiality of assets. In this case, data is not damaged, but it is made available to people who should not have access to it. An example of a threat to confidentiality is when a credit card database held by a company is stolen, with the potential for illegal use of card information.
2. Threats to the integrity of assets. These are threats where systems or data are damaged in some way by a cyberattack. This may involve introducing a virus or a worm into software or corrupting organizational databases.
3. Threats to the availability of assets. These are threats that aim to deny use of assets by authorized users. The best-known example is a denial-of-service attack that aims to take down a website and so make it unavailable for external use.

To counter these threats, organizations should put controls in place that make it difficult for attackers to access or damage assets. It is also important to raise awareness of cybersecurity issues so that people know why these controls are important and so are less likely to reveal information to an attacker.

Examples of controls that may be used are:

1. Authentication, where users of a system have to show that they are authorized to access the system. The familiar login/password approach to authentication is a universally used but rather weak control.
2. Encryption, where data is algorithmically scrambled so that an unauthorized reader cannot access the information. Many companies now require that laptop disks are encrypted. If the computer is lost or stolen, this reduces the likelihood that the confidentiality of the information will be breached.
3. Firewalls, where incoming network packets are examined, then accepted or rejected according to a set of organizational rules. Firewalls can be used to ensure that only traffic from trusted sources is allowed to pass from the external Internet into the local organizational network.

15.2 List the benefits of software reuse and explain why the expected lifetime of the software should be considered when planning reuse.

- Accelerated development

Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.

- Effective use of specialists

Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.

- Increased dependability

Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed.

- Lower development costs

Development costs are proportional to the size of the software being developed. Reusing software means that fewer lines of code have to be written.

- Reduced process risk

The cost of existing software is already known, while the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is especially true when large software components such as subsystems are reused.

- Standards compliance

Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.

If you are developing a long-lifetime system, you should focus on the maintainability of the system. You should not just think about the immediate benefits of reuse but also of the long-term implications. Over its lifetime, you will have to adapt the system to new requirements, which will mean making changes to parts of the system. If you do not have access to the source code of the reusable components, you may prefer to avoid off-the-shelf components and systems from external suppliers. These suppliers may not be able to continue support for the reused software. You may decide that it is safer to reuse open-source systems and components as this means you can access and keep copies of the source code.

16.4 Why is it important that components should be based on a standard component model?

Component-based development embodies good software engineering practice. It often makes sense to design a system using components, even if you have to develop rather than reuse these components. Underlying CBSE are sound design principles that support the construction of understandable and maintainable software:

1. Components are independent, so they do not interfere with each other's operation. Implementation details are hidden. The component's implementation can be changed without affecting the rest of the system.
2. Components communicate through well-defined interfaces. If these interfaces are maintained, one component can be replaced by another component providing additional or enhanced functionality.
3. Component infrastructures offer a range of standard services that can be used in application systems. This reduces the amount of new code that has to be developed.

17.9 List the benefits that a distributed component model has when used for implementing distributed systems.

The five benefits of developing systems as distributed systems:

1. Resource sharing

A distributed system allows the sharing of hardware and software resources—such as disks, printers, files, and compilers—that are associated with computers on a network.

2. Openness

Distributed systems are normally open systems—systems designed around standard Internet protocols so that equipment and software from different vendors can be combined.

3. Concurrency

In a distributed system, several processes may operate at the same time on separate computers on the network. These processes may (but need not) communicate with each other during their normal operation.

4. Scalability

In principle at least, distributed systems are scalable in that the capabilities of the system can be increased by adding new resources to cope with new demands on the system. In practice, the network linking the individual computers in the system may limit the system scalability.

5. Fault tolerance

The availability of several computers and the potential for replicating information means that distributed systems can be tolerant of some hardware and software failures. In most distributed systems, a degraded service can be provided when failures occur; complete loss of service only occurs when there is a network failure.



Thanks for attention!

