COMPUTER SECURITY AND CYBER LAW (CSCL)

LECTURER: ROLISHA STHAPIT/ DIKSHYA SINGH

UNIT 5

Design Principles and Common Security related programming problems LH 4

- Eight principles for the design and implementation of security mechanisms, Common Security related programming problems (Improper choice of initial protection domain, Improper Isolation of implementation detail, Improper change, Improper Naming, Improper de-allocation or deletion, Improper validation, Improper indivisibility, Improper Sequencing, Improper choice of operand or operation).

Basic Principles for Design and Implementation of Security

- Saltzer and Schroeder describe eight principles for the design and implementation of security mechanisms. The principles draw on the ideas of simplicity and restriction.
- **Simplicity** makes designs and mechanisms easy to understand and less can go wrong with simple designs. Minimizing the interaction of system components minimizes the number of sanity checks on data being transmitted from one component to another.
- **Restriction** minimizes the power of an entity. The entity can access only information it needs. Entities can communicate with other entities only when necessary, and in as ways as possible.

Principles for the Design and Implementation of Security Mechanisms

1. Principle of Least Privilege

- **The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task.**
- If a subject does not need an access right, the subject should not have that right. Furthermore, the function of the subject (as opposed to its identity) should control the assignment of rights.
- If a specific action requires that a subject's access rights be augmented, those extra rights should be relinquished immediately on completion of the action.
- This is the analogue of the "need to know" rule: if the subject does not need access to an object to perform its task, it should not have the right to access that object. More precisely, if a subject needs to append to an object, but not to alter the information already contained in the object, it should be given append rights and not write rights.
- This principle restricts how privileges are granted. This principle requires that processes should be confined to as small a protection domain as possible.

2. Principle of Fail-Safe Defaults

- **The principle of fail-safe defaults states that, unless a subject is given explicit access to an object, it should be denied access to that object.**
- This principle requires that the default access to an object is none.
- Whenever access, privileges, or some security related attribute is not explicitly granted, it should be denied.
- Moreover, if the subject is unable to complete its action or task, it should undo those changes it made in the security state of the system before it terminates. This way, even if the program fails, the system is still safe.

3. Principle of Economy of Mechanism

- **The principle of economy of mechanism states that security mechanisms should be as simple as possible.**
- This principle simplifies the design and implementation of security mechanisms.
- If a design and implementation are simple, fewer possibilities exist for errors.
- The checking and testing process is less complex, since fewer components and cases need to be tested.
- Complex mechanisms often make assumptions about the system and environment in which they run and incorrect assumptions results in security problems.

4. Principle of Complete Mediation

- **The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed.**
- This principle restricts the caching of information, which often leads to simpler implementations of mechanisms.
- Whenever a subject attempts to read an object, the operating system should mediate the action.
- First, it determines if the subject is allowed to read the object.
- If so, it provides the resources for the read to occur. I
- f the subject tries to read the object again, the system should check that the subject is still allowed to read the object.

5. Principle of Open Design

- This principle suggests that complexity does not add security.
- **The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation**.
- Designers and implementers of a program must not depend on secrecy of the details of their design and implementation to ensure security.
- Others can search out such details either through technical means, such as disassembly and analysis, or through nontechnical means, such as searching through garbage receptacles for source code listings (called "dumpster-diving").
- If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism. The term "security through obscurity" captures this concept exactly.

6. Principle of Separation of Privilege / Separation of Duty

- This principle is restrictive because it limits access to system entities.
- **The principle of separation of privilege states that a system should not grant permission based on a single condition.**
- This principle is restrictive as it limits access to system entities.
- This principle is equivalent to the separation of duty principle, the principle of separation of duty states that if two or more steps are required to perform a critical function, at least two different people should perform the steps.
- Example: Company checks for more than $75,000 must be signed by two officers. If either does not sign, the check is not valid. The two conditions are the signatures of both officers.

7. Principle of Least Common Mechanism

- This principle is restrictive because it limits sharing.
- **The principle of least common mechanism states that mechanisms used to access resources should not be shared.**
- This principle is restrictive because it limits sharing.
- Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized.
- Isolation prevents communication.
- Communication with something, another process or a resource, is necessary for a breach of security.
- Limit the communication, means we limit the damage. So, if two processes share a resource, by coordinating access, they can communicate by modulating access to the entire resource.

8. Principle of Psychological Acceptability

- This principle recognizes the human element in computer security.
- **The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.**
- Configuring and executing a program should be as easy and as intuitive as possible, and any output should be clear, direct, and useful.
- If security-related software is too complicated to configure, system administrators may unintentionally set up the software in a non secure manner.
- Similarly, security-related user programs must be easy to use and must output understandable messages.
- If a password is rejected, the password changing program should state why it was rejected rather than giving a cryptic error message. If a configuration file has an incorrect parameter, the error message should describe the proper parameter.

Common Security Related Programming Problems

- Unfortunately, programmers are not perfect. They make mistakes.
- These errors can have disastrous consequences in programs that change the protection domains.
- Attackers who exploit these errors may acquire extra privileges (such as access to a system account such as root or Administrator).
- They may disrupt the normal functioning of the system by deleting or altering services over which they should have no control.
- They may simply be able to read files to which they should have no access.
- So the problem of avoiding these errors, or security holes, is a necessary issue to ensure that the programs and system function as required.

A. Improper Choice of Initial Protection Domain

- Flaws arise from incorrect setting of permissions or privileges
- There are three objects for which permissions need to be set properly: the file containing the program, the access control file, and the memory space of the process.

1. Process Privileges

□The principle of least privilege dictates that no process have more privileges than it needs to complete its task, but the process must have enough privileges to complete its task successfully

□ **Management Rule**

- Check that the process privileges are set properly

□ **Implementation Rule**

-Structure the process so that all sections requiring extra privileges are modules. The modules should be as small as possible and should perform only those tasks that require those privileges

2. Access Control File Permissions:

- Biba‘s model emphasize that the integrity of the process relies on both the integrity of the program and integrity of the access control file.

- Verifying the integrity of the access control file is critical, because that file controls the access to role accounts.

- **Management Rule**

The program that is executed to create the process, and all associated control files, must be protected from unauthorized use and modification. Any such modification must be detected.

- **Implementation Rule**

Ensure that any assumptions in the program are validated. If it is not possible, document them for the installer and maintainers, so they know the assumptions that attackers will try to invalidate.

3. Memory Protection

- Consider sharing memory. If two subjects can alter the contents of memory, then one could change data on which the second relies. This sharing poses a security problem because the modifying process can alter variables that control the action of the other process. Each process should have a protected, unshared memory space.

- **Management Rule**

Configure memory to enforce the principle of least privilege. If a section of memory is not contain executable instructions, turn execute permission off for that section of memory. If the contents of a section of memory are not altered make that section read-only.

- **Implementation Rule**

Ensure that the program does not share objects in memory with any other program and that other programs cannot access the memory of a privileged.

4. Trust in the System

- The program relies on the system authentication mechanisms to authenticate the user and on the user information database to map user and roles into their corresponding UIDs.

- It also relies on the inability of ordinary users to alter the system clock.

- **Implementation Rule**

Identify all systems components on which the program depends. Check for errors whenever possible, and identify those components for which error checking will not work.

B) Improper Isolation of Implementation Detail

- The problem of improper isolation of implementation detail arises when an abstraction is improperly mapped into an implementation detail.
- Consider how abstractions are mapped into implementations. Typically, some function (such as a database query) occurs, or the abstraction corresponds to an object in the system. What happens if the function produces an error or fails in some other way, or if the object can be manipulated without reference to the abstraction.
- **Implementation Rule**: The error status of every function must be checked. Do not try to recover unless the cause of the error, and its effects, do not affect any security considerations. The program should restore the state of the system to the state before the process began, and then terminate.

1. **Resource Exhaustion and User Identifiers**

- The designers and implementers decided to have the program fail if, for any reason, the query failed. This application of the principle of fail-safe defaults ensured that in case of error, the users would not get access to the role account.

2. **Validating the Access Control Entries**

-The expression of the access control information is therefore the result of mapping an abstraction to an implementation.

-The question is whether or not the given access control information correctly implements the policy.

3. **Restricting the Protection Domain of the Role Process**

C) Improper Changes

- This category describes data and instructions that change over time. The danger is that the changed values may be inconsistent with the previous values.

1. Memory

- Any process that can access shared memory can manipulate data in the memory. Unless all processes that can access the shared memory implementation a concurrent protocol for managing changes one process can change data on which a second process relies

- This could cause the second process to violate the security policy.

- **Implementation Rule**

  - If a process interacts with other processes, the interactions should be synchronized. In particular, all possible sequences of interactions must be known and for all such interactions, the process must enforce the required security policy.

- **Implementation Rule**

  - Asynchronous exception handlers should not alter any variables except those that are local to the exception handling module. Exception handler should block all the exception

- **Implementation Rule**

  - Data that the process trusts and the data that it receives from un-trusted sources should be kept in separate areas of memory.

D) Improper Naming

- Improper naming refers to an ambiguity in identifying an object.
- Most commonly, two different objects have the same name. The programmer intends the name to refer to one of the objects, but an attacker manipulates the environment and the process so that the name refers to a different object.
- Avoiding this flaw requires that every object be unambiguously identified. This is both a management concern and an implementation concern.
- **Management Rule:** Unique objects require unique names. Interchangeable objects may share a name.
- **Implementation Rule:** The process must ensure that the context in which an object is named identifies the correct object.

E) Improper Deallocation or Deletion

- Failing to delete sensitive information raises the possibility of another process seeing that data at a later time.
- Cryptographic keywords, passwords and other authentication information should be discarded once they have been used.
- As a consequence of not deleting sensitive information is that dumps of memory, which may occur if the program receives an exception or crashes for some other reason.
- **Implementation Rule**

  - When the process finishes using a sensitive object, the object should be erased, the de-allocated or deleted. Any resources not needed should also be released.

F. Improper Validation

- This problem arises when data is not checked for consistency and correctness.
- The process can check correctness only by looking for error codes or by looking for patently incorrect values.

1. **Bounds Checking**

- Errors of validation often occur when data is supposed to lie within bounds.
- For example, a buffer may contain entries numbered from 0 to 99. If the index used to access the buffer elements takes on a value less than 0 or greater than 99, it is an invalid operand because it accesses a nonexistent entry.
- The variable used to access the element may not be an integer (for example, it may be a set element or pointer), but in any case it must reference an existing element.
- **Implementation Rule:**

- Ensure that all array references access existing elements of the array. If a function that manipulates arrays cannot ensure that only valid elements are referenced, do not use that function. Find one that does, write a new version, or create a wrapper.

2.Type Checking

- Failure to check types is another common validation problem. If a function parameter is an integer, but the actual argument passed is a floating point number, the function will interpret the bit pattern of the floating point number as an integer and will produce an incorrect result.
- **Implementation Rule**: Check the types of functions and parameters.
- **Management Rule**: When compiling programs, ensure that the compiler flags report inconsistencies in types. Investigate all such warnings and either fix the problem or document the warning and why it is spurious.

3. Error Checking

- Third common problem involving improper validation is failure to check return values of function.
- **Implementation Rule**

  - Check all function and procedure executions for errors.

4. Checking for valid, not invalid data.

•This principle requires that valid values be known and that all other values be rejected. Unfortunately, programmers often check for invalid data and assume that the rest is valid.

•**Implementation Rule**

  - Check that a variable's values are valid.

•Management Rule

  - If a trade-off between security and other factors results in a mechanism or procedure that can weaken security, document the reasons for the decision, the possible effects, and the situations in which the compromise method should be used.

5. Checking Input

- All the data from the un-trusted sources must be checked
- **Implementation Rule**

- Check all user input for both form and content. In particular, check integer for values that are too big or too small, and check character data for length and valid characters. (check that a variable's values are valid.)

6. Designing for validation

- Sometimes data cannot be validated completely.
- **Implementation Rule**

- Create data structure and function in such a way that they can be validated.

G. Improper Indivisibility

- It arises when an operation is considered as one unit in the abstract but is implemented as two unit.
- **Implementation Rule**

- If two operations must be performed sequentially without an intervening operation, use a mechanism to ensure that the two cannot be divided.

H) Improper Sequencing

- Means that operations are performed in an incorrect order.
- **Implementation Rule**

- Describe the legal sequences of operations on a resource or object. Check that all possible sequence of the programs involved match one legal sequences.

I. Improper Choice of Operand or Operation

- Preventing errors of choosing the wrong operand or operation requires that the algorithms be thought through carefully (to ensure that they are appropriate).
- At the implementation level, this requires that operands be of an appropriate type and value, and that operations be selected to perform the desired functions.
- The difference between this type of error and improper validation lies in the program. Improper implementation refers to a validation failure.
- The operands may be appropriate, but no checking is done. In this category, even though the operands may have been checked, they may still be inappropriate.
- **Management Rule:**

- Use software engineering and assurance techniques (such as documentation, design reviews, and code reviews) to ensure that operations and operands are appropriate.