Information Security

Assignment # 01

Principles of Secure Design

Mam Hina Binte Haq

Course Instructor
CS- 3002
SE- S

Due Date: Feb 17, 2023

Group Members:

Zeeshan Ali	20i-2465
Ans Zeeshan	20i-0543

Assignment # 01

Q1: Show a code example of each of the following principles, i.e. one piece of code that follows each principle and one piece of code that does not:

1- Principle of Least Privilege:

Followed:

The Code that follows the principle of least privilege restricts access to resources or functionality to only what is necessary for the code to perform its intended task. This reduces the risk of unauthorized access or unintended consequences resulting from a bug or malicious code.

```
def read_file(filename):
    with open(filename, 'r') as f:
        return f.read()
# Only read access is required, so file is opened in read-only mode
file_contents = read_file('important_file.txt')
```

In this example, the read_file function takes a filename as input and returns the contents of the file. The code follows the principle of least privilege by opening the file in read-only mode, which restricts access to only what is necessary to read the contents of the file.

Violation:

Code example that does not follow the principle of least privilege:

```
def read_file(filename):
    with open(filename, 'w') as f:
        f.write('some data')
# Unnecessary write access is granted, which can result in unintended consequences
read_file('important_file.txt')
```

In this example, the read_file function takes a filename as input and writes some data to the file. The code does not follow the principle of least privilege because it grants write access to the file, which is unnecessary for the intended task of reading the file. This can result in unintended consequences, such as overwriting the contents of the file or introducing a security vulnerability.

2- Principle of Fail-safe defaults:

Followed:

Code that follows the principle of fail-safe defaults assumes that all inputs are potentially malicious and takes appropriate defensive measures to prevent unintended consequences or security vulnerabilities resulting from invalid input.

Code example that follows the principle of fail-safe defaults:

```
def divide (a, b):
  if b == 0:
      return None
   else:
      return a / b
# Defensive check is added to prevent division by zero
result = divide (10, 2)
if result is not None:
   print(result)
```

In this example, the divide function takes two numbers as input and returns the result of dividing the first number by the second number. The code follows the principle of fail-safe defaults by adding a defensive check to prevent division by zero, which can result in an exception or undefined behavior.

Violation:

Code example that does not follow the principle of fail-safe defaults:

```
def divide (a, b):
   return a / b
# No defensive check is added to prevent division by zero, which can result in an exception or undefined behavior
result = divide (10, 0)
```

In this example, the divide function takes two numbers as input and returns the result of dividing the first number by the second number. The code does not follow the principle of fail-safe defaults because it does not add a defensive check to prevent division by zero. This can result in an exception or undefined behavior, which can lead to unintended consequences or security vulnerabilities.

3- Principle of Economy of Mechanism:

Followed:

Code example that follows the principle:

```
function validatePassword(password)
{
    return password.length >= 8 && /[a-z]/.test(password)
            && /[A-Z]/.test(password) && /[0-9]/.test(password);
}
```

In this code, the function validatePassword checks whether a given password meets certain criteria for being strong. The implementation follows the principle of economy of mechanism because it uses a simple and efficient approach to validate the password by checking the length of the password and the presence of lowercase, uppercase, and numeric characters.

Violation:

Code example that does not follow the principle:

```
function authenticateUser(username, password)
{
     const user = getUserByUsername(username);
      if (user && user.password === password)
       {
          return user;
       }
  return null;
}
```

In this code, the function authenticateUser checks whether a given username and password match the ones stored in the system. The implementation does not follow the principle of economy of mechanism because it exposes the password as plain text in the system. An attacker who gains access to the system can easily read the passwords and use them for malicious purposes. A more secure implementation would use hashing and salting to protect the passwords from being easily compromised.

4- Principle of Complete Mediation:

Followed:

Code example that follows the principle:

```
function withdrawMoney(account, amount)
{
  if (account.isAuthenticated() && account.getBalance() >= amount)
 {
    account.setBalance(account.getBalance() - amount);
    return true;
  }
  return false;
}
```

In this code, when a user wants to withdraw money, the function withdrawMoney checks whether the user is authenticated and has sufficient balance to withdraw the amount. This function follows the principle of complete mediation because it checks every time whether the user is authenticated and has enough balance before performing the withdrawal.

Violation:

Code example that does not follow the principle:

```
function grantAccess(user)
 {
  user.accessLevel = "admin";
}
```

In this code, the function grantAccess gives the user an "admin" access level without any mediation. This function does not follow the principle of complete mediation because it does not check if the user has the right to be granted the "admin" access level. This can cause a security vulnerability if a user who is not authorized gains access to sensitive information.

5- Principle of Separation of Privileges:

Followed:

Code example that follows the principle:

```
function deleteUser(user, isAdmin) {
  if (isAdmin) {
    deleteUserFromDatabase(user);
  }
}
```

In this code, the function deleteUser allows an administrator to delete a user from the database. The implementation follows the principle of separation of privileges because it requires the isAdmin parameter to be set to true before allowing the deletion of a user. This way, regular users do not have the ability to delete other users from the system.

Violation:

Code example that does not follow the principle:

```
function readData() {
  const data = getDataFromDatabase();
  return data;
}
```

In this code, the function readData reads data from the database without any access control. This function does not follow the principle of separation of privileges because it allows anyone who has access to the system to read the data from the database. This can lead to unauthorized access to sensitive information.

6- Principle of least common mechanism:

Followed:

Here's an example of code that follows the Principle of Least Common Mechanism:

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount
    def get_balance(self):
        return self.balance
account1 = BankAccount(1000)
account2 = BankAccount(500)
account1.deposit(500)
account2.withdraw(200)
```

In this example, we have a simple implementation of a bank account. The BankAccount class has three methods - deposit, withdraw, and get_balance. Each account has its own balance, which can only be accessed through these methods. The Principle of Least Common Mechanism is enforced here because each account has its own instance of the balance attribute. This means that the balance of each account is not shared between different objects, and any changes to the balance of one account will not affect the balance of any other account.

Violation:

On the other hand, here's an example of code that violates the Principle of Least Common Mechanism:

```
class SharedBankAccount:
    balance = 0
    @classmethod
    def deposit(cls, amount):
        cls.balance += amount
    @classmethod
    def withdraw(cls, amount):
        if amount > cls.balance:
            raise ValueError("Insufficient funds")
        cls.balance -= amount
    @classmethod
    def get_balance(cls):
        return cls.balance
account1 = SharedBankAccount()
account2 = SharedBankAccount()
account1.deposit(1000)
account2.withdraw(500)                (ruby)
```

In this example, we have a class SharedBankAccount that represents a bank account. However, instead of each account having its own instance of the balance attribute, the balance attribute is shared between all instances of the class. This violates the Principle of Least Common Mechanism, because any changes to the balance of one account will affect the balance of all other accounts.

For example, when we deposit 1000 into account1, the balance is updated to 1000. However, when we withdraw 500 from account2, the balance is updated to -500, which is not what we intended. This demonstrates how sharing a common mechanism (the balance attribute) can lead to unexpected behavior and security vulnerabilities.

Question#02:

1. Hardware security module:

It follows the **Principle of Least Common Mechanism**. It has limited access to the network through a moderated interface that is strictly controlled by internal rules. The principle states that mechanisms used to access resources should not be shared which is being achieved by Hardware security module. It gives isolation and security to the information.

2. Cuckoo sandbox for malware analysis:

It follows the **Principle of Least Common Mechanism**. Cuckoo sandbox is a tool that launches malware in a secure and isolated environment. If that environment gets infected it doesn't affect the user's system as principle states that mechanisms used to access resources should not be shared.

It also follows **Open Design** principle as the developers of Cukoo sandbox has kept it open source so the errors and faults can be discovered and resolved.

3. Access control list in an operating system:

It follows the principles of **Fail Default** as according to it unless a user is given access to a particular file/task, the user access to that file is denied by default.

Principle of Least Privileges is also in the access control list as the access to the users is restricted/limited.

Principle of Complete Mediation is also enforced as every time a user/group tries to access a file the access control list checks if the user is allowed access to that file or not.

4. Image Captcha on Flex:

Principles being followed in its design is **Principle of Separation** states that access shouldn't be granted based on single condition only and the Image captcha also checks another condition so that bots can be filtered out of Flex website and human/legitimate users are granted access only.

5. Password strength indicator on Google or similar websites when you create an account:

The principle being followed here is the **Principle of Open Design**. As the indictor on Google is constantly and explicitly showing that whether the password is strong or weak. It provides guidelines about what the password is lacking so that the user select a password that is secure and strong which will help in case there is an attack.

6. Biometric authentication required before using banking app:

There are two principles being followed in its design: **Principle of Least Common Mechanism** and **Principle of Economy of mechanism**. Former one states that the user is uniquely identified as passwords we set for a particular app can also be used in another app hence common passwords are used but using biometric enforces the LCM as is is unique for every person. As far as economy of mechanism is concerned the biometric authentication is a very simple secure and easy way of authentication.

7. The way encryption ciphers like AES were designed:

It follows the **Open Design Principle** as it does depends upon the secrecy of its implementation and follows a standardized algorithm which is used universally. So in case of any vulnerability the flaw can be detected easily.

8. Atomicity in database transactions:

It follows the **fail default** mechanism as the atomicity assures that the transaction process is occurred completely in any other case it rolls back the transaction.

9. Intrusion detection systems in front of public facing servers of an organization:

It follows the **Principle of Complete Mediation**. Inbound and outbound traffic is monitored and if suspicious activity is observed, then this detection system immediately reports the owner that something is not right, and it requires action. Thus it detects whether a user is a legitimate user or an attacker.

Question #03:

Go through the CVE at the following link:

https://www.cve.org/CVERecord?id=CVE-2007-0408

Explain which principal violation is represented by this CVE.

Ans:

"Although reusing cached connections, BEA WebLogic Server 8.1 through 8.1 SP4 does not correctly validate client certificates, allowing remote attackers to gain access through an untrusted X.509 certificate."

It represents the violation of the Principle of Complete Mediation. When giving access, WebLogic Server must determine whether the user is eligible. The principle is violated because this check is not performed "every time" (it is performed just occasionally).

Question #04:

Explain how the principle of least common mechanism violated or enforced in the following scenarios:

1. **Air-gapping of important machines/servers:**

An air gap is a security measure that includes separating a computer or network and preventing it from connecting to the outside world. Because access to resources is not widely shared, the principle of Least Common Mechanism is being **applied** here.

2. **Cloudflare protection for websites:**

Cloudflare decreases website touch points by directing incoming traffic to Cloudflare servers, which can block malicious activity. Cloudflare **enforces** the Principle of Least Common Mechanism in this manner.

3. **Colonial Pipeline Ransomware attack:**

This attack was carried out by a phishing email sent to one of the company's workers. The attack would not have extended to many other computes in the organization if the Principle of Least Common Mechanism had been followed. As a result, this assault **violates** the Least Common Mechanism Principle.

Work Division:

Name	Roll No	Work Division
Zeeshan Ali	20i-2465	Question 1 & 3
Ans Zeeshan	20i-0543	Question 2 & 4