Defensive Coding & CodeQL Integration with GitHub Repositories

In the **previous lab**, we learned about **CodeQL** and **basic defensive coding techniques** to identify and remediate security vulnerabilities.

Today's focus:

- We will **practice more defensive coding techniques**.
- We will also **integrate a GitHub repository with CodeQL** to automatically detect security issues in projects.

1. Identifying and Remediating Security Vulnerabilities

The following code snippets contain **security vulnerabilities** that students must analyze, explain, and fix using **secure coding practices**.

(a) Insecure Authentication Logic

🟥 **Insecure Code:**

```
if (IsValidUsername($username) == 1) {
    if (IsValidPassword($username, $password) == 1) {
        print "Login Successful";
    }
    else {
        print "Login Failed - incorrect password";
    }
}
else {
    print "Login Failed - unknown username";
```

🚨 **Issues:**

1. **User Enumeration:** The error messages allow attackers to determine if a username exists.
2. **Improper Condition Handling:** The else statement placement can cause logical errors.
3. **Weak Password Handling:** It lacks secure password hashing and comparison.

☑ Secure Code:

```
if (IsValidUsername($username) && IsValidPassword($username, $password)) {
    print "Login Successful";
} else {
    print "Login Failed"; // Generic message to prevent user enumeration
}
```

🔓 **Fixes:**

- Used **generic failure messages** to prevent user enumeration.
- Improved **logical structure** of the conditions.
- Assumes **passwords are securely hashed and validated**.

(b) Logging Sensitive Information

🟥 **Insecure Code:**

```
import logging
logging.basicConfig(level=logging.INFO)
def get_credit_card_number(username):
    if is_authorized_user(username):
        query = "SELECT ccn, expiry_date FROM userCreditCardDetails WHERE username = ?"
        # Execute query (simulated)
        ccn = "1234-5678-9012-3456"
        expiry_date = "12/26"
        # ❌ Insecure Logging (Exposes Sensitive Data)
        logging.info(f"username: {username}, CCN: {ccn}, Expiration Date: {expiry_date}")
        return ccn
    return "Unauthorized"

def is_authorized_user(username):
    return True  # Simulating authorization check
# Example Usage
```

```
print(get_credit_card_number("JohnDoe"))
```

🚨 **Issues:**

1. **Sensitive Data Exposure:** Logs contain **credit card numbers (CCN) and expiration dates**.
2. **Lack of Masking:** The full CCN is returned without masking.
3. **SQL Query Execution Missing Prepared Statement:** Though not visible, the method could be vulnerable to **SQL injection** if not handled properly.

☑ **Secure Code:**

```
import logging
logging.basicConfig(level=logging.INFO)
def get_masked_credit_card(username):
    if is_authorized_user(username):
        query = "SELECT ccn FROM userCreditCardDetails WHERE username = ?"
        # Execute query (simulated)
        ccn = "1234-5678-9012-3456"
        masked_ccn = mask_credit_card(ccn)
        # ☑ Secure Logging (No Sensitive Data)
        logging.info(f"username: {username} accessed masked credit card info.")
        return masked_ccn
    return "Unauthorized"
def mask_credit_card(ccn):
    return "****-****-****-" + ccn[-4:]
def is_authorized_user(username):
    return True  # Simulating authorization check
# Example Usage
print(get_masked_credit_card("JohnDoe"))
```

🔓 **Fixes:**

- **Removed sensitive data from logs** to comply with security regulations (e.g., PCI-DSS).
- **Masked the CCN**, returning only the last 4 digits.
- **Used secure logging practices** to avoid data leakage.

🛠 Student Task: Secure File Handling and Input Validation

Task Overview:

The following Python script contains **insecure file handling, weak input validation, and improper exception handling**. Your task is to:

1. **Analyze the insecure code** and identify vulnerabilities.
2. **Fix the vulnerabilities** using secure coding practices.
3. **Explain what you fixed and why.**

```
import os
def read_file():
    filename = input("Enter the filename to read: ")
    # Insecure: No validation of filename, allowing path traversal attacks
    with open(filename, "r") as file:
        print(file.read())
def delete_file():
    filename = input("Enter filename to delete: ")
    # Insecure: Directly deleting without checking if it's safe
    os.remove(filename)
    print(f"{filename} deleted successfully!")
def execute_command():
    command = input("Enter shell command: ")
    # 🚨 Insecure: Allows arbitrary command execution
    os.system(command)
if __name__ == "__main__":
    print("Options: 1) Read File  2) Delete File  3) Execute Command")
    choice = input("Enter option (1/2/3): ")
    if choice == "1":
        read_file()
    elif choice == "2":
        delete_file()
    elif choice == "3":
        execute_command()
    else:
        print("Invalid choice")
```

Task 2: Integrate CodeQL with GitHub to Detect Security Vulnerabilities

Question:
Set up CodeQL analysis on a GitHub repository, run automated security scans, and generate a detailed vulnerability report. Your report should include:

1. **Steps followed** to integrate CodeQL with GitHub Actions.
2. **List of detected vulnerabilities** in the project.
3. **Analysis of at least three critical vulnerabilities**, including their impact and secure fixes.
4. **Screenshots of CodeQL setup and security alerts**.
5. **Link to your fixed GitHub repository** with security improvements.

Submit a **PDF report** with all the required details.