Secure Patterns and Anti-Patterns



Dimitar Rebranin

College of Engineering and Technology

Utah Valley University

10/10/2024

IT-6780-701 – Secure Coding

Prof. Kodey Crandall

Table of Contents

[1 Introduction 4](#_Toc179409102)

[2 Secure Pattern and Anti-Pattern 1 4](#_Toc179409103)

[2.1 Anti-Pattern 1: Hard Coding Sensitive Information 4](#_Toc179409104)

[2.2 Secure Pattern 1 4](#_Toc179409105)

[3 Secure Pattern and Anti-Pattern 2 4](#_Toc179409106)

[3.1 Anti-Pattern 2 4](#_Toc179409107)

[3.2 Secure Pattern 2 5](#_Toc179409108)

[4 Secure Pattern and Anti-Pattern 3 5](#_Toc179409109)

[4.1 Anti-Pattern 3 5](#_Toc179409110)

[4.2 Secure Pattern 3 5](#_Toc179409111)

[5 References 5](#_Toc179409112)

# 1 Introduction

In software design and development, secure patterns and anti-patterns help guide best practices for building secure systems and avoiding common security pitfalls. Secure patterns are best practices that enhance the security of software systems. They aim to protect against common vulnerabilities and ensure robust defense mechanisms. Anti-patterns are common but flawed approaches that lead to security vulnerabilities. These should be avoided in secure system design.

# 2 Secure Pattern and Anti-Pattern 1

## 2.1 Anti-Pattern 1: Hard Coding Sensitive Information

* **Description**: Storing sensitive information like API keys, passwords, certificates, and even tokens in plain text in the code.
* **Impact**: Hardcoding this type of information exposes it to potential security risks. This data could be shared by accident or pushed to third party repository like Git or Stash. Could be shared with other teams in multiple environments or leaked in the event of data breach. It also becomes difficult to update the secrets, passwords, etc.
* **Example**: Password in plain text

# Password to CMDB

pass='Password123'

## 

## 2.2 Secure Pattern 1: Use environment variables

* **Proposed Solution**: Store sensitive information in environmental variable.
* **Implementation**: Request the password when the script is executed.

# Enter your password for CMDB securely

read -s -p "Enter CMDB Password:" pass

* **Benefits**: Sensitive data is kept outside the source code which will reduce the risk of exposure.

# 3 Secure Pattern and Anti-Pattern 2

## 3.1 Anti-Pattern 2: Not Using HTTPS

* **Description**: This happens when we use HTTP instead of HTTPS for communication. Using HTTP fails to secure the connection, and it doesn’t encrypt the data transmitted between the client and server.
* **Impact**: An attacker can easily eavesdrop on the network traffic and capture the sensitive information. The result of that could lead to altering the data in transit which can corrupt the information.
* **Example**:

<form action="http://dimitar.rebranin/login" method="post">

    <input type="text" name="username" />

    <input type="password" name="password" />

</form>

## 3.2 Secure Pattern 2: Use HTTPS for Secure Communications

* **Proposed Solution**: Use HTTPS to encrypt communication between the client and server.
* **Implementation**:

<form action="https://dimitar.rebranin/login" method="post">

<input type="text" name="username" required />

<input type="password" name="password" required />

</form>

* **Benefits**: The data in transit will be encrypted, protecting it from eavesdropping and interception. Prevents attackers from tampering with the data and provides some level of trust and credibility, as usually there is a visual indicator like lock symbol in the upper left of the browser bar.

# 4 Secure Pattern and Anti-Pattern 3

## 4.1 Anti-Pattern 3: Poor Session Management

* **Description**: Using insecure session mechanisms, such as exposing session IDs in URLs or not regenerating sessions IDs after authentication.
* **Impact**: Attackers can hijack the sessions and use them to login straight to your account and gain unauthorized access.
* **Example**:

<?php

session\_start();

// Insecure session ID handling

if (!isset($\_SESSION['user\_id'])) {

    $\_SESSION['user\_id'] = $\_GET['user\_id']; // Assigning user ID from URL, no validation

}

// No session expiration mechanism

echo "Welcome, User ID: " . $\_SESSION['user\_id'];

// No secure flag on session cookies

setcookie("PHPSESSID", session\_id(), 0, "/");

?>

## 4.2 Secure Pattern 3: Secure Cookies and HttpOnly

* **Proposed Solution**: Ensure session cookies are only transmitted over HTTPS and are no accessible via JS.
* **Implementation**:

setcookie("PHPSESSID", session\_id(), [

    'secure' => true,     // Only allow the cookie over HTTPS

    'httponly' => true,   // Prevent access from JavaScript

    'samesite' => 'Strict'// Prevent cross-site cookie usage

]);

* **Benefits**: HttpOnly flag prevents client-side scripts from accessing cookies, mitigating XSS attacks. The secure flag ensures cookies are only transmitted over encrypted HTTPS connections.

# 5 References

[1] CloudRaft. (2023, June 17). *Secure coding best practices*. <https://www.cloudraft.io/blog/secure-coding-best-practices>

[2] OpenAI. (2023). *ChatGPT* (Mar 14 version) [Large language model].<https://chat.openai.com/chat>