THE USE OF SECURE DESIGN PATTERNS IN OBJECT-ORIENTED PROGRAMMING

9.4HD – Research Project

DUC MANH NGUYEN

8/1/25

COS20007 – Object-Oriented Programming

Table of Contents

I. Abstract:

This short research will walk you through how secure design patterns can make software more secure and reliable while implementing Object-oriented programming (OOP). This research's goal is to see how well common patterns (Singleton, Proxy and Factory) can enhance the level of protection and fix security issues like invalid input, authentication and control access. During the research process, I used mixed-methods approach that include manually comparing, looking at open-sources code and looking at anti-patterns. With the help of coding tools like IDE-based code and a rubric for evaluating design patterns, the results will show how design principles can make real-world applications more secure. At the end, this study is in line with the unit learning outcomes of COS20007 unit because it helps me develop practical skills and analysis.

II. Introduction

Design patterns are widely used to improve software structure, but their role in enhancing security is often overlooked. This research focuses on how secure OOP patterns - such as Singleton, Proxy, and Factory which can be applied to address key vulnerabilities in software systems. The motivation behind this study is to explore how thoughtful design choices can prevent issues like improper input handling, weak authentication, and access violations. By understanding both secure and insecure design approaches, the project aims to highlight the importance of pattern selection in building robust applications. This topic not only deepens understanding of OOP principles but also aligns with the COS20007 unit by strengthening technical and analytical skills.

III. Methods

1. Experimental Comparision: With & Without Secure Desgin Patterns

This method involves the implementation and comparison of two simple code versions: one that follows a secure design pattern approach and one that does not. The purpose is to empirically demonstrate how secure OOP patterns can enhance security and maintainability. The same functional requirements will be used for

both versions, ensuring the only variable is the design pattern usage. The secure version will incorporate the Singleton, Proxy, and Factory patterns to handle common tasks such as object creation, access control, and interface exposure. Meanwhile, the insecure version will use straightforward, pattern-less procedural code.

Both versions will be evaluated using static code analysis tools within an integrated development environment (Visual Studio 2022), to detect common vulnerabilities. Manual code inspection will also be conducted to assess clarity, modularity, and security-relevant concerns.

Specific security-related tasks—such as restricting access to sensitive components, validating input, and enforcing consistent object instantiation—will be used to test the robustness of each approach. Rubric will be applied to score key design and security criteria, such as modularity, clarity, reusability, and vulnerability resistance.

The relevant codes are submitted throught Canvas with the name Method1_[class name].

Here is a brief picture of the main program:

...\Bill\Desktop\COS20007\SecurityResearchLab\Program.cs 1

```
1  namespace SecurityResearchLab
2  {
3      public class Program
4      {
5          static void Main(string[] args)
6          {
7              Console.WriteLine("-= Secure Login Test =-");
8              Credential userInput = CredentialFactory.Create();
9              IDataAccess data = new SecureProxy(userInput);
10
11             data.AccessSensitiveData();
12         }
13
14     }
15
16     //Without Secure Desgin
17     //namespace LoginSystemInsecure
18     //{
19     //    class Program
20     //    {
21     //        static string storedUsername = "admin";
22     //        static string storedPassword = "1234";
23
24     //        static void Main(string[] args)
25     //        {
26     //            Console.WriteLine("=== Insecure Login System ===");
27
28     //            Console.Write("Enter username: ");
29     //            string user = Console.ReadLine();
30
31     //            Console.Write("Enter password: ");
32     //            string pass = Console.ReadLine();
33
34     //            if (user == storedUsername && pass == storedPassword)
35     //            {
36     //                Console.WriteLine("Login successful!");
37     //                Console.WriteLine("Accessing sensitive data...");
38     //            }
39     //            else
40     //            {
41     //                Console.WriteLine("Invalid credentials.");
42     //            }
43     //        }
44     //    }
45     //}
46
47 }
48
```

This method allows for direct, side-by-side comparisons and helps visualize how design decisions affect real-world software security.

2. Pattern Identification and Analysis in Open-Source Projects

This method focuses on examining how secure design patterns are implemented in real-world open-source object-oriented programming projects. The goal is to observe the frequency, correctness, and effectiveness of pattern usage in addressing common software security concerns. Projects will be selected from platforms such

as GitHub, prioritizing C# based repositories with active maintenance, substantial codebases, and relevance to application-level development.

To be more specific, here I used the following GitHub repositories to use as research examples:

1. https://github.com/aspnetboilerplate/aspnetboilerplate
2. https://github.com/blowdart/AspNetAuthorizationWorkshop
3. https://github.com/aspnet/Security

Using code search tools and pattern recognition characteristics, the result will identify instances of design patterns like Singleton, Proxy, and Factory within these projects. Code segments implementing these patterns will be isolated and analyzed to assess their role in preventing security flaws such as unauthorized access or lack of input validation.

This method not only provides insight into best practices but also highlights potential gaps in real-world development. Projects that misuse or fail to apply secure design patterns will be documented to provide counterexamples. A design evaluation rubric will be used to score the effectiveness of pattern usage.

This analysis provides a contextual understanding of the adoption (or lack thereof) of theoretical design principles in real-world development environments. By comparing textbook applications of design patterns with their frequently unstructured real-world counterparts, it also fosters critical thinking. The results will help identify the advantages and disadvantages of adopting secure patterns in various development contexts.

3. Comparison of Secure Design Patterns and Anti-Patterns

This method involves a critical comparison between secure design patterns and anti-patterns - common but flawed design approaches that can introduce or expose software vulnerabilities. The aim is to understand not only how secure patterns prevent security issues but also how anti-patterns can unintentionally lead to them.

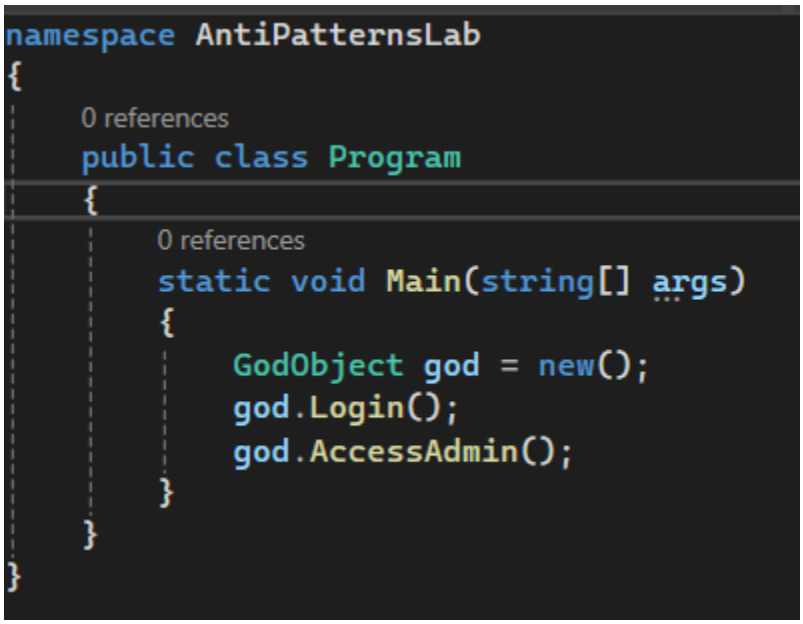
Anti-patterns such as God Object, Spaghetti Code, and Double-Checked Locking will be reviewed by analysing simple C# code that represent their basic characterisitcs. Their points will be contrasted with those of secure patterns like Singleton, Proxy, and Factory, with specific emphasis on how each approach handles object instantiation, input validation, access control, and coupling.

Visual Studio will be used to write sample anti-pattern implementations. Static code analysis tools, vulnerability scanning plugins, and manual inspection under the guidance of a pattern evaluation rubric will all be used to assess the security implications of each example. Aspects like readability, maintainability, vulnerability to security vulnerabilities, and conformity to OOP principles will all be scored by the rubric.

This approach highlights the real-world repercussions of bad design decisions and enables a deeper comprehension of "what not to do." By demanding a critical eye and attention to detail, it also reinforces learning objectives linked to analyzing and debugging code.

The relevant codes are submitted throught Canvas with the name Method3_[class name].

Here is a brief picture of the main program:



IV. Results

1. Method 1: Experimental Comparision

- The secure version (which applied secure design patterns) had fewer vulnerabilities and errors with clearer structure.

- Static code analysis flagged input handling and access control issue on the “insecure” one.

- Rubric evaluation show the secure version score higher on: Modulariy, Resusabilty and Security Resilience.

2. Method 2: Open-source Pattern Identification

- Singleton and Factory patterns were commonly used.

- Proxy was less commonly implemented.

- Code with well-implemented patterns often are: Better in access controlling and Higher maintainability.

3. Method 3: Anti-Patterns

- Anti-patterns like God Object and Spaghetti Code led to:

  - Poor readability
  - High coupling
  - Security exposure

- Secure versions using patterns were:

  - Easier to maintain
  - More secure against input/access issues

- Static analysis showed anti-patterns increased risks, while patterns reduced them.

- Rubric results confirmed higher scores for pattern-based solutions in:

  - Clarity
  - Maintainability
  - Security robustness

V. Discussion

In my opinion, methods 1 and 2 strongly support that secure design patterns improve both code quality and system performance. The structured version using Singleton, Proxy and Factory performance significantly better in all aspects than

the insecure version and anti-patterns examples (God Object – which include too many roles and functions into it without logical plans). These practical outcomes confirm that design patterns are effective tools in the security field.

Regarding method 2, it revealed the gap between theoretical knowledge and real-world implementation. The result suggests that coder benefits the most from secure design strategies. In overall, the results emphasize that desgin patterns are not just for improve the performance of coding project or making it clearer but it also very critical in the security aspects of big projects later. This aligns perfectly with the course ULO which is to build reliable systems and coding projects.

VI. Conclusions

This research discussed how design patterns that are most applied in security aspects (Singleton, Proxy, and Factory), can be utilized to enhance software security in OOP. Experimental comparison, actual project analysis, and anti-pattern inspection all together showed that employing secure patterns leads to more maintainable, modular, and secure code. Insecure approaches and anti-patterns were shown to increase vulnerability and complexity.

These findings confirm the importance of using methodical design at an early phase of the development lifecycle to prevent common software security flaws. Future research can examine automatic identification of secure pattern applications in large systems or examine additional patterns (e.g., Observer, Strategy) in different programming languages. A study of how teams use and maintain secure design habits over very long-time spans in agile, or DevOps environments would also be informative on further advancing long-term code quality and security practices.