A Practical Approach to Infuse Security Patterns into Undergraduate Software Engineering Course Projects

[1]Ingrid A. Buckley

[1]Computer Science Department, Tuskegee University
Tuskegee, Alabama 36830, USA

Abstract

This paper introduces the use of security patterns as a teaching aid to help software engineering students better understand and solve security problems. With the wide spread use of various applications in different domains and industries security is a growing problem that requires swift address. The lack of knowledge about security and vulnerabilities is a primary problem facing society today. It is important to educate future software engineers from early on, by introducing them to practical methods to solve security problems. Introducing security patterns in the software engineering projects offer students hands-on experience and exposure in solving security problems. The objective is to show the relationship between the different levels of the the engineering process to develop applications showing how various design and security patterns can be integrated. A comprehensive security focused software engineering outline is provided that can be adapted for different software engineering projects with a strong emphasis on security.

Keywords: *Software Engineering, Security Patterns. Design Pattern, Security, Secure System Design.*

1. Introduction

Software systems are continuously becoming more advanced and integrated in society to perform personal, business, educational, and scientific needs. Our heavy reliance on these systems requires that we find effective ways to build and evaluate their performance to avoid issues and problems. Several security breaches [2, 3] have already been experienced that have affected millions of customers. These breaches have led to identity theft and millions of dollars to correct the vulnerabilities. Software developers are charged with developing sound and secure systems for everyday use. One common problem with this important responsibility is having the knowledge and experience in developing secure systems, since most developers are experts in the core principles of programming but are not security experts.

This aim of this approach is to aid, train and educate software developers from early on, on how to build secure systems. Software engineering is concerned with the principles and theories which embody the creation, maintenance and evolution of different systems. Security awareness and exposure can contribute greatly to how current and future software engineers and developers approach, solve and build the systems we have come to depend on day to day. Security and design patterns provide the knowledge; approach and guidelines that will help software developers learn best practices and solutions to build secure systems.

Section 2 provides motivation and challenges faced by software developers and by extension software engineering students who will develop applications. Section 3 presents background information on security breaches, and security patterns. Section 4 presents the structure of most software programs, the primary software development steps, and design and security patterns that can be applied at each stage. Section 5 provides a security-focused outline that can be adapted for different software engineering projects in a software engineering course and section 7 concludes.

2. Motivation

Computer Science majors are required to take the Software Engineering course which introduces students to the process of designing and implementing programs. This course is very heavy in the sense that it has a lot of information about various software engineering methods, as well as a semester long project that has to be implemented. The goal is to infuse security and design patterns in the software engineering projects to help student implement programs which have security built in throughout the entire process of software development. Security patterns are useful in teaching because they provide

proven solutions to security problems. It can be cumbersome and frustrating for students to solve security problems effectively in one semester where they have to learn the various ins and outs of software engineering in addition to completing the development of a major project.

3. Background

Security has been of great concern for some time, and efforts to address the various security issues have increased, as technology evolves and as society become increasingly more reliant on it. To set the tone of this paper we will first learn about some security breaches that have negatively impacted businesses and customers worldwide. Additionally a brief summary is provided highlighting the purpose and advantages of using security patterns.

3.1 Security Breaches

In November 2013 the Target store databases were attacked, the personal and credit card information of 40 million customers were compromised [2]. In April 2010, hackers gained access to approximately 77 million PlayStation Network accounts. In this attack unencrypted credit card numbers, personal information and purchase history were compromised [3]. RSA servers were compromised by hackers which is also the security division of EMC which is a huge storage company used by many financial institutions. EMC stores close to 40 million authentication tokens used by employees to access corporate and government networks, the hackers were able to gain access to these tokens. Since this incident, EMC spent over 60 million to monitor the information of concerned clients. Similarly in August 2007, hackers attacked Monster.com and stole resume information of 1.3 million job seekers [3]. These incidents are common and hackers continue to strengthen their efforts in attacking corporate, e-commerce and government systems. The problem associated with security breaches are far reaching and affect other aspects such as privacy and reliability. Security patterns can be used in software engineering/development solutions to solve security problems [8].

3.2 Security Patterns

Patterns [4] embody the experience and knowledge of many designers and when properly catalogued, they provide a repository of solutions for useful problems. Initially used for improving code, patterns are becoming a staple tool to build secure systems [6, 7]. The POSA [11] template defines a systematic way to describe patterns. It consist of approximately eleven units, each describes one aspect of a pattern. This template is designed to capture the experience and knowledge of professionals that have solved common problems. Patterns support best practices. Each unit of the POSA Template is described below:

1. **Name** - the name of the pattern should correspond to the generic name given to the specific type of attack in standard attack repositories such as CERT [12].
2. **Intent or thumbnail description** - A short summary of the intended purpose of the pattern, including which problem it solves.
3. **Example** of a specific problem.
4. **Context** -this section describes where the pattern applies, including prerequisites and the general environment.
5. **Problem** - describe the forces which affect the solution, attacks.
6. **Solution** - describes the general idea of solving the problem, it includes UML models (static and dynamic), formalization.
7. **Implementation** – provides recommendations and hints for implementers
8. **Example resolved** - describes how the pattern solved the specific problem
9. **Known uses** - provides at least three examples of use in real systems
10. **Consequences** – provides advantages and disadvantages of the pattern's solution.
11. **Related patterns** - presents complementary or alternative patterns.

Security patterns describe mechanisms that control threats. Security patterns join the extensive knowledge accumulated about security with the structure provided by patterns to provide guidelines for secure system construction and evaluation. Security has had a long trajectory, resulting in a variety of approaches to analyze security problems and to design security mechanisms. It is helpful to capture this expertise in the form of patterns [11]. There are several books [4] on security patterns and academic institutions that write and share security patterns. An attacker can attack a system from all levels. If a hacker does not find vulnerability in level n, then level n+1 or n-1 may have vulnerabilities that can be exploited. It is important to identify attacks at every level or stage in software development. Security patterns provide the following advantages:

- Security patterns embody experience and good design practices.
- They help to prevent errors, and save time.
- Can be reused.
- Provide guidelines to solve security problems.

- Provide best case solutions to common problems.

4. Infusing Patterns in Software Engineering

It is important to at some core security and design patterns that can be applied to different security problems at different stages in the software development process.

4.1 Software Engineering Process

There are several approaches to software development; the most common are plan-driven, agile and incremental development. Each present their strengths and weakness and is usually selected based on the type of system being developed. Though they are different, they all involve some fundamental software engineering principles which are done at different times depending on the software approach being used. This approach focuses on the core units which cannot be avoided in software development and map those to come useful patterns.

4.1.1 Requirements

Eliciting Requirements/Specifications begins the initial phase of software development thus it is important for students to capture and precisely define accurate software requirements. Accurate software requirements can be achieved by insisting that students define, discuss and present these requirements to the instructor and other stakeholders who may be involved with the project. This will help students refine and include requirements that are overlooked initially. This process is cyclic and is generally updated throughout the entire software development life cycle.

4.1.2 Design

System Design can present a challenge for software engineering students. The use of design patterns can help students find solutions that they may not have the experience and skills to address. Design patterns can speed up the development process, as effective software design requires considering issues that may not become visible until later in implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems later in development. They also improve code readability for programmers who are familiar with the patterns. Some common design patterns [5] include: Observer, Watchdog, Mediator, Façade, Composite, Strategy and Singleton.

4.1.3 Implementation

System Implementation and development with a focus on security presents an even greater challenge for students. The use of security and design patterns serve as a useful guide for them. For example some patterns [9, 10] may also provide examples and known uses in multiple programming languages, including C++, C, and Java. Some common security patterns [10] include: Acknowledgement, Role Base Access Control, Secure Strategy Factory, Secure Builder Factory, Secure Chain of Responsibility, Secure Logger, Clear Sensitive Information and Guarded suspension.

4.1.4 Testing

System Validation and Testing may seem simple for some students and often this exercise is neglected until the last minute. As a result, students may not test their program as rigorously as required because they want to avoid missing their deadline. It is necessary to encourage students to create test cases that test every requirement and its associated tasks as outlined in section 5.0. The test cases must be tested at the unit, component and system level. At the bare minimum, students are instructed to test all possible successful and unsuccessful outcomes at each level.

5. Security-focused Project Guideline

Software Engineering/Development courses can be challenging for undergraduate computer science students due to the heavy theoretical and practical components that must be covered in such courses. In this course, students are introduced to the theoretical side of software engineering by learning the software life cycle concepts, software requirements and specifications, object-oriented design, detailed modular design, validation and verification, proving program correctness, software testing, software quality assurance, and project management. All of these topics are vast and can be taught separately as individual courses, as a result this course involves a lot reading material for students to cover.

Once students have the theoretical background discussed above, they are required to utilize this knowledge in a semester long group project. This group project is comprised of two major units: comprehensive documentation and an implementation of the documentation in the form of a program. It is not practical to cover all of the mentioned topics early enough in the semester for students to develop a project documenting the various steps in software engineering design and then providing an

implementation that reflects that documentation. This feat is even more difficult when security has to be incorporated in every aspect of the group project.

To handle this challenge, a phased weekly approach is used; where students develop their group project as they learn the topics. A weekly project deliverable system is adopted for this particular course, where students have to develop different units of their group project and present them during class to the instructor. This method helps to ensure that students are working on the project and getting helpful feedback instead of waiting until the last minute to hurry through all the necessary software stages.

5.1 Weekly Project Deliverables Outline:

Students are instructed to write the documentation for the group software engineering/development project and adhere to the weekly deliverable outline in detail.

Week 1: Group Project Deliverable-1:
Project Proposal: Part A

- Project Description:
  - Provide a clear and succinct description about the product you want to develop
- Vision:
  - Describe the group's vision for the project and product
- Motivation and Justification:
  - Justify why you want to create the product and its purpose
  - Describe your target group – the users that will use your system

Week 2: Group Project Deliverable-2:
Project Proposal: Part B

- Project Goals and Objective :
  - Describe your general and security goals and objectives
- Project Approach:
  - Describe how you intend to carry out the project
    - Estimate how much time it will take to complete the project
  - Research and describe in detail the resources needed
  - Discuss the technology and languages that will be used
- Team Members:
  - Describe roles and responsibility of each group member
  - Describe the effort and contribution of each group member once the project is complete
  - Specify clearly what each group member was responsible for doing in the project and if the tasks were completed.
  - Specify what percentage (max is 100%) of the overall project was completed by each group member, you must justify how you came up with this percentage

Week 3: Group Project Deliverable-3:
Project Specifications/Requirements: Part A

- Each requirement should be well defined and unambiguous. There are different types of requirements try to categorize them as follows:
  - User requirements, business requirements and system requirements etc.
  - Label each requirement, e.g. **R1, R2….RN.**
- Once the requirements and specifications have been identified for your project, break each of those requirements into smaller units called tasks
  - Label each task, e.g. **T1, T2…TN.**
  - Provide a full description of each task.

Week 4: Group Project Deliverable-4:
Project Specifications/Requirements: Part B

- Continue with the tasks created for each requirement in Part A
- Analyze each task carefully and determine and identify potential threats and vulnerabilities that it may introduce
  - Provide a full description of each threat and vulnerability under each respective task
  - c. Label each threat, e.g. **TH1, TH2….THN**
  - d. Label each vulnerability, e.g. **V1, V2….VN**

Week 5: Group Project Deliverable-5:
Project Specifications/Requirements: Part C

- Continue with the threats and vulnerabilities identified in Part B
- Select the countermeasures/defenses to handle the identified threats and vulnerabilities

- Label each defense, e.g. **D1, D2….DN**
- Justify your reason for choosing a particular countermeasure/defense
    - Include the strengths of the security countermeasure/defense, the challenges they introduce (functionality, speed etc.)
    - Discuss why the security countermeasure/defense was selected and how effective it will be
- Discuss any standards/regulations that may apply to each threat and vulnerability
- If you identify threats that you will not be able to address, justify why,

Week 6: Group Project Deliverable-6:
Project Specifications/Requirements: Part D

- Continue with the countermeasure/defense selected for each threat and vulnerability
- Select security patterns that provide those countermeasure/defense
    - Justify your reason for choosing a particular security pattern
    - Provide the advantages, disadvantages and consequence for each of the security patterns selected
- Select design patterns to implement the requirements/ tasks
    - Justify your reason for choosing a particular design pattern
    - Provide the advantages, disadvantages and consequence for each of the security patterns selected

Week 7: Group Project Deliverable-7:
Project Specifications/Requirements: Part E

- Project Risks, Milestones and Contingencies:
    - Identify any risks associated with the requirements
    - Label each milestone, e.g. **RK1, RK2…RKN.**
    - Provide a full description of each risk.
- Every requirement should be associated with a milestone
    - Label each milestone, e.g. **M1, M2…MN.**
    - Provide a full description of each milestone.
- Determine strategies/contingencies to handle any potential risks and the overall cost of the strategy/contingency.
    - Label each contingency, e.g. **C1, C2…CN**
    - Provide a full description of each contingency.
    - Include a justification for choosing a particular contingency

Project Schedule:

- All group members are expected to participate
- Divide requirements and the tasks among group members
- Provide a project schedule [1]
    - Show each task, the effort needed to complete it in days, the actual duration in days and the dependencies associated with each task
- Create an activity bar chart [1] showing asks over the set time line for the project Create group member allocation chart [1] showing all the tasks, timeline associated with the group member it was assigned to .

Week 8: Group Project Deliverable-8:
Design – UML Diagrams: Part A

- Use the tasks associated with each requirement from previous work to create the project's UML diagrams
- Provide a description for each UML diagram. This description should be comprehensive to help the reader understand each diagram readily.
- All UML diagrams must use the correct UML notation
- Create Use Case diagrams for your program
    - Show all actors, subsystems, functions and scenarios
- Create Activity diagrams
    - Show all activities for each operation/function provided by your system
    - Security Process
        - Examine each activity and determine if there are threats or vulnerabilities that you looked over
        - Repeat week 5 and week 6 to address the new threat or vulnerability.

Week 9: Group Project Deliverable-9:
Design – UML Diagrams: Part B

- Use the tasks associated with each requirement from previous work to create the project's UML diagrams
- Provide a description for each UML diagram. This description should be comprehensive to help the reader understand each diagram readily.
    - Create Class diagrams
        - Show all classes, class associations, generalizations, compositions, aggregations, methods, attributes, multiplicity, and visibility of attributes and methods
    - Create Sequence diagrams
        - Show all the dynamics and interactions between classes and users within your system

Implementation: Part A

- The following units are optional depending on the nature of the project:
    - Design the User Interface (Optional)
        - Create the relevant user interfaces
        - Include an illustration of the user Interface, user Interface drawing
        - Describe each user interface and the function of each component on the UI
        - Discuss the tools, languages used to create the user interface
        - Describe and justify why you selected the design and layout of the user interface
    - Create Database (Optional)
        - Include an Entity Relationship Diagram (ERD) for the database
        - Discuss how your database enforces referential Integrity
        - Discuss the overall database Schema design

Week 10 -12: Group Project Deliverable-10, 11, 12:
Implementation: Part B

- Demo working application of at least two (2) core function for the program weekly until all requirements are implemented
- The core functions must implemented using the design patterns selected previously
- Implementation must include the security patterns selected previously to mitigate the identified threats or vulnerabilities
- Report any challenges or set-backs being experienced
- All code must be fully documented

Validation and Testing: Part A

- For every new function developed describe the techniques used to test the function – Include unit, component and system testing etc.

Week 13: Group Project Deliverable-13:
Implementation: Part C

- Demo working application showing each core functions that is implemented
- Code must be fully documented

Validation and Testing: Part B

- Describe the techniques used during implementation in terms of - unit testing, component, and system testing
- Describe outcome/result of testing, and updates made
- Discuss Recommendations and Improvement for the project

Week 14: Group Project Deliverable-14:
Other Documentation:

- Maintenance and Evolution:
    - Suggest ways to maintain, enhance and expand your project
- User Manual – provide clear instructions to the user explaining how to use your product.

Week 15: Group Project Deliverable-15:
Conclusion:

- Deliverables
    - Were the specifications, goals, objectives met?
    - Targets and deadlines met?
    - Challenges?
        - Such as unforeseen incidents that impacted the project
        - Discuss anything that could not be completed
- Describe the end product
- References – remember to include all references

Week 16: Group Project Deliverable-16:
Assessment Committee:

- The group will present their final project to a committee for evaluation
- The final project documentation must be sent to the evaluation committee at least two (2) days before the final group presentation

- The evaluation committee will interview the group as a whole
- Each group member will be interviewed separately by the evaluation committee

5.2 Evaluation

Instructors can choose to ask students to select a project of their choosing or provide a list of instructor selected projects. If the instructor chooses a project, some of the group project deliverables in week1 can be adjusted.

5.3.1 Advantages:

Assessing group effort can present a challenge for an instructor in terms of evaluating the individual effort of each group member. The weekly project deliverable approach allows the instructor to assess the performance of each group member because each deliverable leading up to the final project is presented in class and graded. As the course progresses, the interaction between group members and the instructor during presentations will allow the instructor to properly assess individual student effort and contribution. Additionally, an evaluation committee can also provide feedback to the instructor after the final debut of the completed project.

5.3.2 Challenges:

Time is a major challenge in teaching software engineering courses. Often students complain that they will not have time to complete their project or may propose to implement a fraction of the intended functionalities, or to not complete the implementation portion of the project at all. In such instances, students can receive partial credit if they can develop a working prototype of the software project. Prototypes are useful to show stakeholders that some effort was invested in the project. Prototypes can be a useful alternative for students who are unable to complete the full scope of a project on time. Another challenge that may be experienced is some students may complain that they do not have the programming skills to implement the design and security patterns and or the basic functions of the system. It is important to test the programming knowledge and skill level of students from early on. It is useful to assign a programming assignment using the relevant languages, tools and concepts that are required for the software engineering course project in order to properly assess the skill level of the students before the group project implementation begins. This will allow the instructor the opportunity to make changes or craft a project that is commensurate with the skill level of the students if necessary.

6. Conclusion

This paper presents a comprehensive outline that can be adopted in Software Engineering and Software Development courses with a concentration in Security. These two courses are often principal courses in most Computer Science programs worldwide. The objective is to introduce security concepts and expose future software developers to security solutions to build secure systems. This approach utilizes security and design patterns, integrated with the standard software engineering approaches that begin system development with requirements.

References

[1] I. Sommerville. (2011). Software Engineering - 9th Edition. Boston, MA: Addison Wesley.
[2] CNN Money. (2013, December).Target: 40 million credit cards compromised. Retrieved from: http://money.cnn.com/2013/12/18/news/companies/target-credit-card/. Last Accessed: 1/30/2014.
[3] CNN Money. (2011, April). 9 of the worst security breaches ever. Retrieved from: http://money.cnn.com/galleries/2012/technology/1206/gallery.9-worst-security-breaches.fortune/2.html. Last Accessed: 1/30/2014.
[4] M. Schumacher, E. B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad.(2006). Security Patterns: Integrating security and systems engineering. West Sussex, England: Wiley Series on Software Design Patterns.
[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides.(1994). Design patterns: elements of reusable object-oriented software. Boston, Ma :Addison-Wesley.
[6] E. B. Fernandez, S. Mujica, and F. Valenzuela.(2011). Two security patterns: Least Privilege and Secure Logger/Auditor. Proceedings of Asian PLoP.
[7] I. A. Buckley and E.B. Fernandez.(2011). Enumerating Software Failures to Build Dependable Distributed Applications. High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium. 120 - 123. doi:10.1109/HASE.2011.35.
[8] I. A. Buckley and E. B Fernandez. (2009).Three patterns for fault Tolerance. Proceedings of the OOPSLA MiniPLoP.
[9] H. Schmidt and J. Jürjens. (2011). Connecting Security Requirements Analysis and Secure Design Using Patterns and UMLsec. 23rd International Conference, CAiSE 2011. 367-382.

London, UK:Springer Berlin Heidelberg. doi:10.1007/978-3-64

[10] D. Schmidt, M. Stal, H. Rohnert, F.Buschmann (2000). Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. Irvine , CA:John Wiley & Sons.

[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. (1996). *Pattern-Oriented Software Architecture: A System of Patterns,* vol. 1, Wiley.

[12] CERT.(1988). Cybersecurity Engineering. Retrieved from: https://www.cert.org/about.Last Accessed: 3/2/2014.

Dr. Ingrid Buckley earned a B.S. (2004) degree in Computing and Information Technology from the University of Technology, Jamaica. She holds a M.S. (2008) and a Ph.D. (2012) from Florida Atlantic University. In 2013 she joined the faculty of Tuskegee University in her current rank of Assistant Professor of Computer Science. Previously, she held a visiting Assistant Professor of Computer Science position at Illinois Wesleyan University. Dr. Buckley's areas of specialization are security, secure software engineering, and database management systems. She has authored six peer reviewed publications, one book chapter and one journal paper. Her current work in software engineering focuses on training and educating undergraduate computer science majors on designing and implementing secure applications. She also trains and educates business and information technology majors on developing relational database models for business operations. Dr. Buckley is a member of the National Society of Black Engineers (NSBE), Upsilon Pi Epsilon and Golden Key International honorary societies.