2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing

A Natural Classification Scheme for Software Security Patterns

Aleem Khalid Alvi
School of Computing
Queen's University
Kingston, Canada
aleem@cs.queensu.ca

Mohammad Zulkernine
School of Computing
Queen's University
Kingston, Canada
mzulker@cs.queensu.ca

Abstract—**Software security patterns are a proven solution for recurring security problems. Security pattern catalogs are increasing rapidly. This creates difficulty in selecting appropriate software security patterns for a particular recurring security problem. There are several classification schemes to organize software security patterns. Every classification scheme has unique selection criteria for choosing a security pattern. However, no classification scheme considers security flaws, which is the root cause of software security vulnerabilities. In this paper, we provide a natural classification scheme for software security patterns. Our classification scheme is associated with software lifecycle phases. Security flaws are incorporated in the classification of software security patterns with security objectives in the requirement phase, security properties in the design phase, and attack patterns in the implementation phase. Furthermore, we enhance the existing security pattern template with classification parameters.**

Keywords: design patterns, software security patterns, pattern classification, secure system development.

I. INTRODUCTION

Design pattern is one of the artifacts, which provide easy and fast implementation of software features. They capture expert knowledge and provide domain independence with reusability. In the area of software security, the subset of design patterns that support security requirements are recognized as software security patterns. Security patterns are used to achieve desired security objectives. They provide ease of use to software developers, who may not have deep understanding of security.

Many security pattern catalogs are introduced for helping developers to use the security patterns in software development [17, 18, 22, 26, 45]. The increase in the number of software security patterns invites a problem in the selection process. The selection of appropriate security patterns for fulfilling a security loophole is a complex problem. Researchers develop the classification of software security patterns based on a number of different perspectives, such as applicability, enterprise architectural spaces, desirable properties, security objectives, threat modeling, software lifecycle phases, text classification, and attack patterns [4-14]. However, none of them uses security flaws to classify security patterns. A security flaw may cause harmful effect on any activity of a software system. Therefore, an application will be vulnerable, if it possesses one or more security flaws.

In this paper, a classification scheme for software security patterns is proposed. The classification scheme of security patterns is divided according to software lifecycle phases. Security flaws are incorporated in the classification of software security patterns with security objectives in the requirement phase, security properties in the design phase, and attack patterns in the implementation phase. These parameters are naturally associated with the software lifecycle phases. The selection of these parameters is adopted by considering the security concepts and their relationships in light of security violation steps [15]. Existing classifications of software security patterns and their corresponding templates are analyzed and a security pattern template is introduced for the classification. A security pattern template stores the information of a software security pattern with the relevant classification parameters for helping in the classification process. The proposed classification associates software flaws to security patterns with the help of security objectives, security properties, and attack patterns. The main objective of this classification is to prevent security flaws from the very beginning of the development process. Our classification approach is unique as it uses security flaws as a classification parameter, which is the origin of any vulnerability exploitations in software applications. The classification will help identify appropriate software security patterns corresponding to security flaws. Therefore, security loopholes caused by security flaws can be prevented easily by using security patterns.

The rest of the paper is organized as follows. In Section II, we describe the related work. Section III depicts the concept, classification parameters, and the proposed classification structure of security patterns. Section IV proposes the security pattern template for storing security pattern information. In Section V, we describe the use of the security pattern classification. This section explains the classification of software security patterns in software lifecycle phases. Finally, in Section VI, we summarize our work and discuss open issues for future research.

II. RELATED WORK

A number of different software security patterns are presented in the literature. Yoder and Barcalow [18], Kenzle and Elder [16], Romanosky [17], and Schumacher *et al.* [2] propose 7, 29, 8, and 46 security patterns, respectively. Heyman *et al.* [19] report that 10 and 35 percent of the 220 reported security patterns in the literature are process activities and guidelines, respectively, and 55 percent of the

220 security patterns can be regarded as "core patterns" (i.e., they fulfill authors' definition of a pattern) [19].

Cheng *et al.* [20] provide a comparison of 8 patterns based on Viega and McGraw's [21] design guidelines. Halkidis *et al.* [5] compare 13 most common software security patterns from the most comprehensive guide developed by Blakley *et al.* [9] to examine the evolution of software security patterns. Laverdiere *et al.* [6] compare 12 software security patterns where most patterns are the same as in the Cheng and Halkidis's comparisons based on the desirable and undesirable properties of patterns.

Kienzle *et al.* [16, 22] classify all security patterns into two broad classes: structural and procedural. Cheng *et al.* [20] describe the classification based on the access types of security patterns and categories into creational, structural, and behavioral patterns. It is called "classification by approach". Further, security patterns are defined in terms of application-level, host-level, and network-level. It is called "classification by abstraction". Blakley *et al.* [9] describe broader classification for security patterns by categorizing the security patterns into two classes based on applicability: available and protected system patterns. Halkidis *et al.* [5] use the work done by Blakley *et al.* [9] for the classification of security patterns based on applicability.

Trowbridge *et al.* [11] introduce a tabular classification scheme for patterns. The table is used for organizing security patterns built on four pillars: the Zachman Framework for Enterprise Architecture [23], the Architectural Standards Description from IEEE 1471-2000 [15], the Enterprise Architecture Framework [24], and the Principles of Test-Driven Development [25]. Laverdiere *et al.* [6] use six-sigma approach for the classification of security patterns using the desirable and undesirable properties. Hafiz *et al.* [8] propose the classification scheme of software security patterns using the CIA (confidentiality, integrity, and availability) model, application context, security wheel, McCumber cube, threat modeling, and hierarchical classification. Further, Hafiz *et al.* [14] classify software security patterns according to attacks through STRIDE[1] threat model.

Yskout and Heyman *et al.* [26] investigate the role played by security patterns in the construction of secure software both from a technical and a methodological perspective. They classify based on two perspectives: development phases and security objectives. Steel *et al.* [27] describe the classification of security patterns based on a tier approach. They classify their patterns according to logical architecture tiers, namely web tier, business tier, web service tier, and identity tier. Yoshioka *et al.* [4] describe security pattern classification based on software lifecycle phases. Sarmah *et al.* [28] use linguistic metaphors and formal concept analysis for a categorization of software security patterns. Fernandez *et al.* [7] use software architecture classification approach to classify architectural patterns addressing the type of concerns. VanHilst *et al.* [29] propose six primary dimensions to classify security patterns independently: lifecycle stage, component source, security response type, architecture layer, constraint level, and domain. Washizaki *et al.* [30] use the work of VanHilst *et al.* [29] and improve it by using dimension and pattern graphs.

Hasheminejad *et al.* [13] propose the text classification of software security patterns and learning techniques. They automate the process for identifying security patterns. Chad *et al.* [46] describe the categorization of secure design patterns using the three classes based on the level of abstraction as follows: the architectural-level patterns, the design-level patterns, and the implementation-level patterns. Wiesauer *et al.* [10] define security pattern taxonomy based on the description of attack patterns, and as a first step, it uses security pattern classification according to attacks through the STRIDE model [14].

III. Overview

Software vulnerability is a security flaw or weakness in system security process, design, or implementation that can be exploited to cause a security violation of the system's security policy [31]. Its existence is a threat for an application, and its exploitation is an attack. It is the primary or natural cause of an attack on software systems.

We use software lifecycle phases for security pattern classification similar to the classification of Yoshioka *et al.* [4] and further classify each phase using software flaws. The schematic diagram of our classification is shown in Fig. 1. According to this classification, security patterns can be divided based on software lifecycle (SLC) phases into the following major areas: (1) security patterns for the requirement phase, (2) security patterns for the design phase, and (3) security patterns for the implementation phase. We use two classification parameters in every software lifecycle phase as shown in Fig. 1. However, we consider security flaws as the main classification parameter since it is the root cause of all software vulnerability. Security flaws exist in every phase of software lifecycle. For example, in the requirement phase, a security flaw may occur because of the missing requirements. In the design phase, it may occur; for example, insufficient buffer size and incomplete or incorrect initialization. In the implementation phase, exceptions may occur because of the programmer mistakes. For example, module interfaces may not be correctly defined or a security module may be missed or incorrectly implemented.

Attacks or security violations are indicators of the presence of security flaws. A group of security control services or countermeasures are deployed to protect a software system against some security violations. These groups of security control services are provided by software security patterns. We consider CIAA (confidentiality, integrity, availability, and accountability) as primary security objectives or properties and the authentication, authorization, non-repudiation, and auditability as secondary security objectives or properties. The relationships between software security patterns and security objectives or properties may be one to one, one to many or many to one. The classification parameters are used as pattern elements in a security pattern template describe in Section IV.

[1] STRIDE is an acronym of six threat categories, i.e., Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

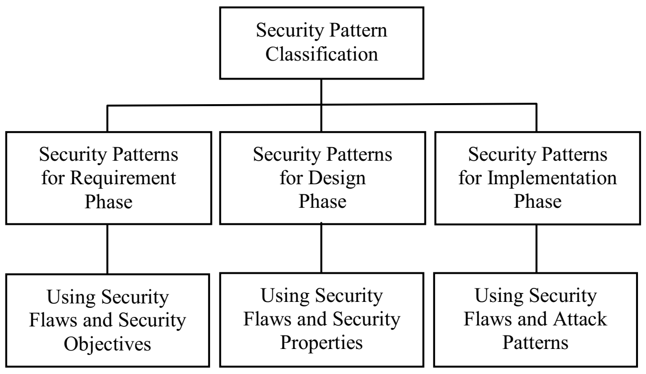
Security Pattern Classification
Security Patterns for Requirement Phase
Security Patterns for Design Phase
Security Patterns for Implementation Phase
Using Security Flaws and Security Objectives
Using Security Flaws and Security Properties
Using Security Flaws and Attack Patterns

Figure 1. Classification Structure of Software Security Patterns

IV. SECURITY PATTERN TEMPLATE

A security pattern template is a document that stores information of a security pattern in a specific format. The first template was introduced by Christopher Alexander *et al.* [3], and was adopted by the Gang of Four in software engineering as design pattern template [39]. A security pattern template is the modification of the design pattern template to fulfill security constraints and requirements. This document is used to find the characteristics of a security pattern to best fit as a solution for the corresponding recurring security problem. We analyze many security pattern templates based on pattern elements and their other details as shown in Table I, and the result of the analysis is used for generating an appropriate security pattern template for the proposed classification by including more pattern elements. Three symbols (▲, •, ▼) are used to show the level of details (from higher to lower level, respectively) of the pattern element's information that are available in the corresponding references of Table I. The absence of a pattern element in the corresponding reference is shown by an empty cell.

In the proposed security pattern template (see Fig. 2), three pattern elements are incorporated to the existing templates [20, 26, 27]: "Security Objectives and Properties, "Related Security Flaws", and "Related Attack Patterns". These fields help developers locate a security pattern corresponding to a security flaw. In the template, pattern elements and its sub-elements (e.g., Solution) are selected more precisely. The security pattern template is used for describing the security pattern classification presented in Section V. Here, we only define the three pattern elements of the proposed template as follows. The definition of the remaining elements can be obtained from the existing literature as listed in Table I.

Security Objectives and Properties (M): It stores information of the main security objectives and properties of the security pattern. Security patterns solve problems by considering these security objectives and properties.

Related Security Flaws (M): It provides the list of related security flaws.

Security Pattern Name (M)

1. Also known as (O)
2. Intent (O)
3. Context and Applicability (M)
4. Security Objectives and Properties (M)
5. Example (O)
6. Problem (M)
    a. Motivation
    b. Forces
7. Solution (M)
    a. Static Structure
    b. Dynamics Structure
    c. Participants
    d. Collaborations
    e. Strategies
8. Implementation (O)
9. Risk and Reality Check (O)
10. Consequences (M)
11. Related Security Flaws (M)
12. Related Attack Patterns (M)
13. Related Design and Security Patterns (M)
14. Known Uses (O)
15. Resulting Context (O)
16. References (O)

Figure 2. Security Pattern Template (M: Mandatory, O: Optional)

Related Attack Patterns (M): It identifies and describes the related attacks and attack patterns.

V. SECURITY PATTERNS IN SOFTWARE DEVELOPMENT PHASES

We explain the proposed classification of software security patterns in detail based on the software development phases in Sections A, B, and C. The pattern elements described in the security pattern template are the parameters of the proposed classification. We use the software flaw taxonomy [42], where the list of possible sources of software flaws is classified based on software lifecycle phases.

We use the software flaws database called Common Weakness Enumeration (CWE)[2]. This is a public database of software weaknesses contributed by many research organizations. The CWE uses a template that has a field called "Time of Introduction". This field provides the information on the possible occurrence of a software flaw or weakness in software lifecycle phases.

A. Security Patterns for the Requirement Phase

In the requirement phase, we consider the potential software flaws from software flaw taxonomy or common weakness enumeration. These software flaws map to the software security patterns using security objectives. A software flaw may be introduced in the requirement phase because of many reasons. For example, software requirements may be ambiguous, inadequate, and obsolete.

[2] http://cwe.mitre.org/index.html

TABLE I. EVOLUTION OF SECURITY PATTERN TEMPLATE

Types	PE / RP	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
APT	[3]	●				▲	▲	▲		▲	▲			▲			●			●		●
DPT	[39]	▲	▲	●	▲				▲			●	●				●	●		●		●
	[40]	●	▲			▲	▲	▲		▲	▲	▼		▲			●			●	▲	●
	[34]	●	▲			▲	▲				▲	▼			▲							
	[32]	●					▲	▲			▲	●										
SPT	[18]	●	▲		▲		▲	▲			▲						▲	●		●		●
	[17]	●	▲		▲		▲	▲			▲							●		●		●
	[41]	●				▲	▲	▲		▲	▲			▲			●			●		●
	[20]	▲	▲	●	▲				▲			▲				▲	●	●		▲		
	[9]	●	▲	●	▲				▲			▲	●				●	●		●		
	[33]	●	▲	▲	●	●	●		●	●	●	▲	▲				●	●		●	●	●
	[27]	●					▲	▲			▲	▲						▲	▲	●		●
	[12]	●		●		▲	▲				▲	▼					●	●		●		
	[26]	●	▲	●	●		▲		▲		▲	▼	▲				●	▲		●		●
	[2]	●	▲			▲	▲				▲	●	●		▲		●	●		▼	●	●
	[45]	●	▲		▲				▲			●	●		▲		●	●				▲

Legends:

Name	A	Forces	G	Resulting Context	M	Related Patterns and Principles	S	Very Detail	▲
Another name	B	Applicability	H	Example Resolved	N	Categorization and References	T	Intermediate Detail	●
Intent	C	Rationale	I	Constraints	O	Example	U	Less Detail	▼
Motivation	D	Solution	J	Known Uses	P	Alexandrian Pattern Template	APT	Pattern Elements	PE
Context	E	Detail parts of solution	K	Consequences	Q	Design Pattern Template	DPT	Research Papers	RP
Problem	F	Implementations	L	Feasibility and Risk	R	Security Pattern Template	SPT		

In other words, customer requirements are captured and changed inappropriately or incorrectly mapped to software requirements. Software security patterns are used to prevent these flaws from occurring in the requirement phase.

We use the security objectives to find software security patterns for the protection against software flaws. The common weakness enumeration uses a template that has a field called "Common Consequences". This field provides a list of security objectives. These security objectives are affected because of the occurrence of software flaws or weaknesses. This information is helpful to find the connection between software flaws and security objectives. The classification uses a template that has a field called "Security Objectives and Properties". In this field, every security pattern stores the supported security objectives and properties information. This field is helpful to connect the security flaws to software security patterns.

In the requirement phase, we consider security patterns based on the analysis and process activities called "analysis process security patterns" and "model-based security patterns" for the categorization of security patterns in the requirement phase [4]. The examples of the "analysis process security patterns" and the "model-based security patterns" are shown in Tables II and III, respectively. We show the relationships between security flaws, security objectives, and security patterns. We associate software flaws to security objectives based on security hunch. Besides, the security flaws and security objectives are used to find the corresponding security patterns through security pattern documentations manually. Every security pattern document stores the information of the related software flaws in the pattern element called "Related Security Flaws" and corresponding affected security objectives in the pattern element called "Security Objectives and Properties", where a security pattern itself is the solution.

Using the classification criteria, three examples for the selection of "analysis process security patterns" are shown in Table II. Every software flaw violates the corresponding security requirements and its objectives. In the third column, a security flaw and the corresponding security objectives from the first and second columns are used to find the security pattern by using security pattern documentations. Security pattern documentation uses security pattern template to write a security pattern with specific information.

There are many types of software security patterns that use different software modeling languages for example, UMLsec, Tropos or Problem Frame in the requirement phase [36]. The patterns using these models are under the category of "model-based security patterns".

TABLE II. EXAMPLES OF SECURITY FLAWS MAPPING TO ANALYSIS PROCESS SECURITY PATTERNS IN THE REQUIREMENT PHASE

Analysis Process Security Patterns		
Security Flaws (Using software flaw taxonomy [42])	**Security Objectives**	**Security Patterns**
Important missing and inaccurate requirements.	Confidentiality, Integrity, Availability, and Accountability.	Build scenario pattern [47].
Improper privacy requirements or privacy related software flaws.	Confidentiality, Integrity, and Accountability.	Privacy process patterns [35]: Authentication, Authorization, Identification, Data Protection, Anonymity, and Pseudonymity patterns.
Any software flaw violating security requirements.	Confidentiality, Integrity, Availability, and Accountability.	Enterprise security and risk management patterns [2]: Security needs identification for assets, Asset valuation, Threat assessment, Vulnerability assessment, Risk determination, and Enterprise security approaches patterns.

Using our classification criteria, three examples for the selection of "model-based security patterns" are shown in Table III. Every software flaw shown in the first column is taken from the common weakness enumeration. In the second column, the security requirements and its objectives are violated by the corresponding software flaws. In the third column, the security flaws and the corresponding security objectives are used to find the security pattern as a security solution manually.

B. *Security Patterns for the Design Phase*

In the design phase, security patterns are classified using security flaws and properties. The selection of a security pattern is a decision based on the design concepts and the detail architecture of the system.

Software security patterns are used for security implementation in the design phase. The list of possible sources for software design flaws indicates the design weaknesses that may invite vulnerabilities in the implementation and operation phases. It is possible that the sources of security problems may occur and develop because of various software design flaws such as unsafe exception handling, wrong algorithmic approach, incorrect data conversion, and inappropriate use of arithmetic and logic decisions [42].

The process of mapping sources of software design flaws from the taxonomy or specific software design flaws from the CWE are selected and associated to security properties based on security hunch. The security design flaws and security properties are used to find the corresponding security patterns through security pattern documentations manually. A proposed security pattern template supports easy searching because of pattern elements: "Security Objectives and Properties" and "Related Security Flaws". In Table IV, we show the results of a process for finding security patterns corresponding to software flaws. This shows the relationship between security flaws, security properties, and software security patterns. In the first column, we use security flaws from the sources of the security flaw taxonomy and the common weakness enumeration. We show the association of the security flaws and security properties to the required security patterns for fulfilling the security requirements in the design phase using security pattern documentation.

Pattern researchers provide the relationships of security properties to software security patterns. For example, Yoder and Barcalow [18] introduce seven conceptual architectural software security patterns for an application security at the system level using a natural language.

Christopher *et al.* [27] develop 23 enterprise security patterns that are pointing architectural use of security patterns for the J2EE enterprise application. The authors map some security patterns to security properties using the pattern element called "Consequences".

TABLE III. EXAMPLES OF SECURITY FLAWS MAPPING TO MODEL-BASED SECURITY PATTERNS IN THE REQUIREMENT PHASE

Model based Security Patterns		
Security Flaws (Using CWE)	**Security Objectives**	**Security Patterns**
CWE-326: Inadequate encryption strength, CWE-261: Weak cryptography for passwords.	Confidentiality.	Cryptography pattern using UMLsec [1].
CWE-807: Reliance on untrusted inputs in a security decision, CWE-296: Improper following of chain of trust for certificate validation.	Confidentiality, Integrity, and Availability.	Monitoring pattern using Secure Tropos for improving delegation and trust [46].
CWE-287: Improper authentication.	Confidentiality, Integrity, and Availability.	Security patterns for securing agent systems using Tropos methodology: Agency guard, Agent authenticator, Sandbox, and Access controller patterns [37].

TABLE IV. EXAMPLES OF SECURITY FLAWS MAPPING TO SECURITY PATTERNS IN THE DESIGN PHASE

Security Flaws (Using security flaw taxonomy [42] and the CWE)	**Security Properties**	**Security Patterns**
Unsafe exception.	Confidentiality and Integrity.	User defined exception pattern [28].
Back door access.	Access verification.	Single access point pattern [18].
Multiple places for user validation.	Authentication and Authorization.	Check point pattern [18].
Broken authentication and Broken access control.	Authentication and Access control.	Assertion builder pattern [27].
Session theft.	Authentication and Identification.	Single sign-on delegator pattern [27].
CWE-266: Incorrect privilege assignment.	Access control.	Roles pattern [18].
CWE-613: Insufficient session expiration.	Auditability.	Session pattern [18].

Schumacher *et al.* [2, 43] describe 25 out of 46 architectural and design software security patterns for addressing the security properties in the design phase. For example, integrity property is obtained by using the "digital signature pattern" or "signed authenticated call pattern" for the voice over IP networks [43]. The availability property is ensured by using firewall patterns and the accountability security property is satisfied by using accounting patterns [2]. The above discussed mappings show that security patterns are used to achieve the required security properties. The proposed security pattern template provides help for pattern miner to easily find the association of related security flaws to security properties and then software security patterns.

C. Security Patterns for the Implementation Phase

For the implementation phase, many researchers develop guidelines for preventing potential security risks [48, 49]. Nevertheless, only guidelines are not adequate to reuse. The patterns consist of semi-structured documents and standard vocabularies are used for encouraging programmers to adopt secure coding techniques during programming [4]. The protection from software flaws may be achieved in the implementation phase by using the implementation-level software security patterns. Software security patterns are used in software system implementations successfully to ensure security.

Security patterns in the implementation phase are classified by using security flaws with the help of attack patterns. Human errors during programming can easily introduce bugs that can be caused vulnerability in the software system. Software flaws may be inherited from the previous software lifecycle phases. Software flaws in many cases occur in more than one phase.

The Common Attack Pattern Enumeration and Classification (CAPEC)[3] is the attack pattern database. The CWE and CAPEC are well connected to each other. Both database schemas use pattern elements to map information from security flaws to attack pattern and vice versa. Similarly, the CAPEC maps the software security patterns to the corresponding attack patterns by using the attack pattern element called "Relevant Security Patterns" [48]. Therefore, common weakness and attack pattern databases are helpful for the classification of software security patterns using security flaws and attack patterns.

We use the list of sources of potential security flaws from [42] and the common weakness enumeration in the implementation phase. In the implementation phase, many expected sources of software flaws are possible. Some examples are legacy code developed for different environment and interfaces, syntax flaws, wrong declaration of data, incomplete link to libraries, and use of untested software. These sources of software flaws may cause security vulnerabilities in software applications. There are many software security patterns available to protect software applications from security vulnerabilities in the implementation phase. In Table V, we show the mapping of the security flaws to attack patterns and use attack pattern database to find the security patterns. First, we use the list of possible sources of software flaws from the taxonomy and the common weakness enumeration. Second, we use the common weakness enumeration to find the attack patterns shown in the software weakness description. The attack patterns are available in the attack pattern database. The database provides the pattern schema description [44]. The schema description provides the attack pattern template element (i.e., Relevant Security Patterns) that stores the information for the corresponding security pattern. However, we find that the pattern element "Relevant Security Patterns" is rarely used in the attack pattern database. This problem can be solved by adopting our security pattern template. Therefore, we search corresponding security patterns manually using the software security patterns documentation and the relevant literature.

A sufficient condition for maintaining security is the prevention of implementation attack (e.g., the buffer overflow attack).We may protect from implementation level software flaws using the security patterns shown in Table V. The "SQL injection" and "cross-site scripting" vulnerabilities are found in many software applications as many software flaws cause the "SQL injection" and "cross-site scripting" attacks. The "intercepting validator pattern" is used for the protection of software applications from these attacks (see Table V). The software flaws corresponding to the "information disclosure", "elevation of privilege", and "spoofing identity" vulnerabilities are shown in the third, fourth and fifth rows of Table V. The "secure proxy", "login tunnel variant", and "container managed security" patterns provide protection from these vulnerabilities and the corresponding attacks.

VI. Conclusion and Future Work

We present a classification scheme for software security patterns based on the natural cause of security breaching in software applications. The prevention of the cause of a security flaw has not been addressed so far by using security patterns. This develops the need of security pattern classification by using security flaws. Therefore, we classify the software security patterns based on security flaws along with other parameters to provide easy use of software security patterns for preventing the root causes of security violations.

We propose the security pattern template based on the analysis of available pattern templates and introduce some classification parameters. We classify software security patterns using software lifecycle phases and each phase is further classified based on security flaws with the help of security objectives in the requirement, security properties in the design, and attack patterns in the implementation phase. In future, we will include more security objectives to achieve finer levels for software security patterns in the classification.

[3] http://capec.mitre.org/

TABLE V. EXAMPLES OF SECURITY FLAWS MAPPING TO SECURITY PATTERNS IN THE IMPLEMENTAIONS PHASE

Security Flaws (Using software flaw taxonomy [42] and the CWE)	**Attack patterns**	**Security Patterns**
Incorrectly filtered escape characters, incorrect type handling, blind SQL injection, parameterized statements, and escaping software flaws, and CWE-89: Improper neutralization of special elements used in an SQL command.	SQL injection attack: CAPEC-108, CAPEC-109, CAPEC-110, CAPEC-66, CAPEC-7.	Intercepting validator pattern [27].
Input invalid or untrusted data, declaration of the data or data structure, and CWE-79: Improper neutralization of input during web page generation.	Cross-site scripting attacks: CAPEC-106, CAPEC-198, CAPEC-199, CAPEC-209, CAPEC-243, CAPEC-244, CAPEC-245.	Intercepting validator pattern [27].
Bypass authentication, CWE-287: Improper authentication, and CWE-263: Password aging with long expiration.	Dictionary attacks, CAPEC-16: Dictionary-based password attack, and CAPEC-114: Authentication abuse.	Secure proxy, Login tunnel variant patterns [38], and Container managed security patterns [27].
Weak or guessable security token (may be password) of the high privilege user, CWE-521: Weak password requirements, and CAPEC-49: Password brute forcing.	Dictionary attack and CAPEC-22: Exploiting trust in client.	Secure proxy, Login tunnel variant patterns [38], and Container managed security patterns [27].
Fudging data and allowing the unprivileged code to use other person identity and CWE-693: Protection mechanism failure.	CAPEC-151: Identity spoofing, CAPEC-195: Principal spoofing, and CAPEC-94: Man in the middle attack.	Secure proxy, Login tunnel variant patterns [38], and Container managed security patterns [27].

ACKNOWLEDGMENT

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would like to thank Dr. Munawar Hafiz of the University of Illinois at Urbana-Champaign, USA for his thoughtful comments on an intial draft of this paper.

REFERENCES

[1] J. Jurjens, “UMLsec: Extending UML for Secure Systems Development,” in *UML 2002—The Unified Modeling Language*, Springer Berlin / Heidelberg, vol. 2460, 2002, pp. 412-425.

[2] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons Ltd., 2006.

[3] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[4] N. Yoshioka, H. Washizaki, and K. Maruyama, “A Survey on Security Patterns,” *Progress in Informatics*, vol. 5, 2008, pp. 35-47.

[5] S. T. Halkidis, A. Chatzigeorgiou, and G. Stephanides, “A Qualitative Evaluation of Security Patterns,” in *Information and Communications Security*, Springer Berlin / Heidelberg, vol. 3269, 2004 , pp. 251-259.

[6] M-A. Laverdiere, A. Mourad, A. Hanna, and M. Debbabi, “Security Design Patterns: Survey and Evaluation,” in *Canadian Conference on Electrical and Computer Engineering*, May 2006, pp. 1605-1608.

[7] E. B. Fernandez, H. Washizaki, N. Yoshioka, A. Kubo, and Y. Fukazawa, “Classifying Security Patterns,” in *Proc. of the 10th Asia-Pacific Web Conference*, China, April 2008, pp. 342-347.

[8] M. Hafiz and E. J. Ralph, “Security Patterns and their Classification Schemes,” University of Illinois at Urbana-Champaign, Technical Report, August 2006.

[9] B. Bob, H. Craig, and et al., “Security Design Patterns,” The Open Group, Technical Report G031, 2004.

[10] A. Wiesauer and J. Sametinger, “A Security Design Pattern Taxonomy based on Attack Patterns,” in *International Joint Conference on e-Business and Telecommunications*, Italy, July 2009.

[11] D. Trowbridge, W. Cunningham, M. Evans, L. Brader, and P. Slater, “Describing the Enterprise Architectural Space,” MSDN, June 2004. http://msdn.microsoft.com/en-us/library/ff648192.aspx.

[12] D. Rosado, C. Gutierrez, E. Fernández-Medina, and M. Piattini, “Defining Security Architectural Patterns Based on Viewpoints,” in *Computational Science and Its Applications*, Springer Berlin / Heidelberg, vol. 4707, 2007, pp. 262-272.

[13] S. M. H. Hasheminejad and S. Jalili, “Selecting Proper Security Patterns Using Text Classification,” in *International Conference on Computational Intelligence and Software Engineering*, 2009, pp. 1-5.

[14] M. Hafiz, P. Adamczyk, and R. E. Johnson, “Organizing Security Patterns,” in *Software, IEEE*, vol. 24, no. 4, pp. 52-60, July 2007.

[15] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Standard 1471-2000, 2000, pp. 1-23.

[16] D. M. Kienzle and M. C. Elder, “Security Patterns for Web Application Development,” Defense Advanced Research Projects Agency, Technical Report F30602-01-C-0164, 2002.

[17] R. Sasha, “Security Design Patterns: Part 1,” Online at CGISecurity, version 1.4, November, 2001. http://www.cgisecurity.com/lib/security DesignPatterns.html. Last Accessed Oct. 2010.

[18] J. Yoder and J. Barcalow, “Architectural Patterns for Enabling Application Security,” in *4th Conference on Pattern Languages of Programs*, 1997.

[19] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, “An Analysis of the Security Patterns Landscape,” in *Proc. of the Third International Workshop on Software Engineering for Secure Systems*, 2007, pp. 3.

[20] B. Cheng, S. Konrad, L. A. Campbell, and R. Wassermann, “Using Security Patterns to Model and Analyze Security Requirements,” in *IEEE Workshop on Requirements for High Assurance Systems*, 2003, pp. 13-22.

[21] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, 1st ed., Addison-Wesley Professional, MA, 2001.

[22] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, “Security Patterns Repository,” version 1.0, 2002. Available: http://www.scrypt.net/~celer /securitypatterns/repository.pdf

[23] J. A. Zachman, “A Framework for Information Systems Architecture,” in *IBM Systems Journal*, vol. 38, no. 2-3, 1999, pp. 454-470.

[24] M. Goodyear, *Enterprise System Architectures: Building Client Server and Web Based Systems*, 1st ed., CRC Press, Sept. 1999.

[25] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed., Addison-Wesley Professional, Nov. 2004

[26] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen, “A System of Security Patterns,” Catholic University of Leuven, Belgium, Dept. of Computer Science, Technical Report CW-469, 2006.

[27] C. Steel, R. Nagappan, and R. Lai, *Core Security Pattern: Best Practices and Strategies for J2EE, Web Services, and Identity Management*, Prentice Hall PTR / Sun Micros, 2005.

[28] A. Sarmah, S. M. Hazarika, and S. K. Sinha, “Security Pattern Lattice: A Formal Model to Organize Security Patterns,” in *International Workshop on Database and Expert Systems Application*, 2008, pp. 292-296.

[29] M. VanHilst, E. B. Fernandez, and F. Braz, “A Multi-dimensional Classification for Users of Security Patterns,” in *Journal of Research and Practice in Information Technology*, vol. 41, 2009, pp. 87-98.

[30] H. Washizaki, E. B. Fernandez, K. Maruyama, A. Kubo, and N. Yoshioka, “Improving the Classification of Security Patterns,” in *20th International Workshop on Database and Expert Systems Application*, 2009, pp. 165-170.

[31] G. Stoneburner, C. Hayden, and A. Feringa, “Risk Management Guide for Information Technology Systems,” Recommendations of the National Institute of Standards and Technology (NIST), Technical Report, NIST Special Publication 800-30, 2002.

[32] P. Avgeriou and U. Zdun, “Architectural Patterns Revisited–A Pattern Language,” in *Proc. of the 10th European Conference on Pattern Languages of Programs*, Irsee, Germany, July 2005.

[33] D. M. Kienzle, M. C. Elder, D. S. Tyree, and J. Edwards-Hewitt, “Security Patterns: Template and Tutorial,” Technical Report, Nov. 2003. http://www.scrypt.net/~celer/securitypatterns/.

[34] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, vol. 1, John Wiley & Sons, England, 1996.

[35] C. Kalloniatis, E. Kavakli, and S. Gritzalis, “Using Privacy Process Patterns for Incorporating Privacy Requirements into the System Design Process,” in *the Second International Conference on Availability, Reliability and Security*, April 2007, pp. 1009-1017.

[36] A. Bandara, H. Shinpei, J. Jurjens, H. Kaiya, A. Kubo, R. Laney, H. Mouratidis, and et al., “Security Patterns: Comparing Modeling Approaches,” The Open University, Department of Computing, Technical Report TR2009-06, 2009.

[37] H. Mouratidis, M. Weiss, and P. Giorgini, “Modeling Secure Systems Using An Agent-oriented Approach and Security Patterns,” in *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, 2006, pp. 471-498.

[38] S. T. Halkidis and E. Chatzigeorgiou, “A Practical Evaluation of Security Patterns,” in *Proc. of International Conference on Artcicial Intelligence and Digital Communications*, 2006, pp. 37-44.

[39] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., Addison-Wesley Professional, 1994.

[40] L. Rising, *The Patterns Handbook: Techniques, Strategies, and Applications*, 1st ed., Cambridge University Press, 1998.

[41] M. Schumacher, “Foundations of Security Patterns,” in *Security Engineering with Patterns*, Springer Berlin / Heidelberg, vol. 2754, 2003, pp. 97-119.

[42] F. Dörenberg, “Dependability Impairments: Faults, Errors and Failures,” in *Analysis and Synthesis of Dependable Electronic Systems*, unpublished, 2003. http://www.nonstopsystems.com/textbook_sample_1.pdf

[43] M. Weiss, “Modelling Security Patterns Using NFR Analysis,” in *Integrating Security and Software Engineering: Advances and Future Visions*, IGI Publishing Hershey, PA, USA, 2006, pp. 127-141.

[44] S. Barnum, “Common Attack Pattern Enumeration and Classification (CAPEC) Schema Description,” Department of Homeland Security, Cigital, Inc., 2008.

[45] D. Chad, S. Kirk, C. S. Robert, S. David, and T. Kazuya, “Secure Design Patterns,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Technical Report CMU/SEI-2009-TR-010, 2009.

[46] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, “Modeling Security Requirements through Ownership, Permission and Delegation,” in *13th IEEE International Conference on Requirements Engineering*, 2005, pp. 167-176.

[47] Y. Liu, J. A. Clark, and S. Stepney, “‘Devices Are People Too’ Using Process Patterns to Elicit Security Requirements in Novel Domains: A Ubiquitous Healthcare Example,” in *Security in Pervasive Computing,* Springer Berlin / Heidelberg, vol. 3450, 2005, pp. 31-45.

[48] T. Shiralkar and B. Grove, “Guidelines for Secure Coding,” atsec information security corporation, Technical Report, January 2009. http://www.atsec.com/downloads/pdf/secure-coding-guidelines.pdf.

[49] K. Hans, “Cutting Edge Practices for Secure Software Engineering,” in *International Journal of Computer Science and Security*, vol. 4, no. 4, 2010, pp. 403-408.