A classification methodology for security patterns to help fix software weaknesses

Loukmen Regainia*, Sébastien Salva† and Cédric Bouhours‡
LIMOS - UMR CNRS 6158
Auvergne University, France
Email: * loukmen.regainia@udamail.fr, † sebastien.salva@udamail.fr, ‡ cedric.bouhours@udamail.fr

Abstract—**Security patterns are generic solutions that can be applied since early stages of software life to overcome recurrent security weaknesses. Their generic nature and growing number make their choice difficult, even for experts in system design. To help them on the pattern choice, this paper proposes a semi-automatic methodology of classification and the classification itself, which exposes relationships among software weaknesses, security principles and security patterns. It expresses which patterns remove a given weakness with respect to the security principles that have to be addressed to fix the weakness. The methodology is based on seven steps, which anatomize patterns and weaknesses into set of more precise sub-properties that are associated through a hierarchical organization of security principles. These steps provide the detailed justifications of the resulting classification and allow its upgrade. Without loss of generality, this classification has been established for Web applications and covers 185 software weaknesses, 26 security patterns and 66 security principles.**

Index Terms*—*software weakness; Security patterns; Security principles; Classification

I. Introduction

Most of the security experts/practitioners admitted that applying the Maginot line syndrome in the software development life cycle (i.e. adding a line of defenses after the development stage) is a bad strategy. Instead, security has to be thoroughly considered throughout the software life cycle. Security at the design stage can be performed by means of security patterns, which are specific patterns helping in designing secure applications. These patterns represent general and reusable solutions to recurring security problems [1]. Since 1997, the number of security patterns is continuously growing and around 170 security patterns are available at the moment [2]. As patterns are often presented with an abstract point of view, they may be differently interpreted according to a given context. As a consequence, the choice of the good security pattern against a security problem is sensitive and somehow perilous [3], [4]. As designers cannot be experts in any field, they clearly lack of guidance during the design phase.

Hence, with the stated goal to guide designers towards good practices, some papers [5] proposed to classify patterns into categories related to security concepts also known as principles, e.g., Authorization, Fault tolerance or Access control. Despite the benefits brought by these studies, this kind of classification remains insufficient because security principles are mostly abstract and lead to imprecise categories from which designers still have to take a decision on the patterns to implement. This is why other works proposed to classify security patterns according to vulnerabilities [6], [4]. For a given security pattern, these classifications provide the vulnerabilities that are mitigated with the application of this pattern. But, they do not precise if several patterns are required. Furthermore, these classifications are established by directly comparing the textual descriptions of patterns with those of the vulnerabilities. As these descriptions are generic and have different levels of abstractions, the categorization of a pattern with a vulnerability can be done only when there is an evident link between both. Many patterns may be lost in the process.

From these observations, we propose another kind of classification of security patterns. But, first and foremost, we propose a methodology of classification, i.e. a list of successive steps, which lead to the classification itself. The primary contributions of this paper can be summarized by the following points:

- we present a classification, which establishes relationships among security weaknesses, security principles and security patterns, expressing which patterns remove a given weakness with respect to the principles that have to be addressed to fix the weakness. We focus on weaknesses provided by the Common Weakness Enumeration (CWE) database because a weakness is an error or a root cause that may lead to a vulnerability in a specific technology, version or language. A weakness represents a known error that a designer wish to avoid;
- our classification reveals the combinations of patterns that should be chosen to remove a given weakness. In addition, our classification also gives the relations among these patterns. Indeed, studies about the inter-pattern relations have been proposed in [7], [8]. We leverage these results and include them in the classification. For instance, these inter-pattern relations offer the advantage to make apparent the conflicting or alternative patterns;
- we propose a methodology, built on seven steps, which anatomizes patterns and weaknesses into sets of more precise sub-properties that are connected by means of a hierarchical organization of security principles. Actually, these steps provide the detailed justifications of the resulting classification. Since patterns and weaknesses are still presented with texts, some of these steps are manually done. But, other steps, in particular the construction of the final classification, are automatically performed. As a consequence, this methodology can be followed again

Research supported by the industrial chair on Digital Confidence (http://confiance-numerique.clermont-universite.fr/index-en.html)

to classify new security patterns or weaknesses, without the need to re-investigate all the patterns and weaknesses considered in the classification.

Without loss of generality, we have limited our classification to the field of Web applications since only for this kind of application, we picked out 185 software weaknesses and 26 security patterns. For the classification, we also organized 66 security principles. In addition, for some weaknesses, we portrayed the classification with Security Activity Graphs (SAG) [9] organizing the principles and patterns related to a weakness in a tree form. This classification is stored in a database available in [10].

The paper is organized as follows: Section II starts by presenting some security pattern classifications and introduce our motivations. In the next section, we recall some security concepts used throughout the paper. We get to the heart of the matter, i.e. the presentation of our classification methodology in Section IV. We give a short presentation of the classification in Section V. We traditionally conclude the paper and outline some perspectives in Section VI.

II. Related Work

The growing number of security patterns available in the literature makes the choice of the most appropriate ones, for overcoming a security problem, very difficult. In order to ease this task, many taxonomies and classifications were proposed in the literature. Alvi et al. outlined 24 of these classifications and established a comparative study to point out their positive and negative aspects [5]. They chose 29 classification attributes (purpose, abstraction levels, life-cycle, etc.) and compared the classifications against a set of desirable quality criteria (navigability, completeness, usefulness, etc.). They observed that several classifications were built in reference to a unique classification attribute, which appears to be insufficient. They indeed concluded that the use of multiple attributes enables the pattern selection in a faster and more accurate manner.

In order to highlight the key challenges in pattern classification, Bunke et al. presented a systematic literature review of the papers dealing with security patterns between 1997 and 2012. In addition, they listed a set of classification criteria and established a comparison between design patterns and security pattern classifications [11]. They finally proposed a classification based upon the application domains of patterns (software, network, user, etc.).

The classifications proposed in [6], [4] give another point of view by helping designers in the choice of patterns to fix software vulnerabilities and weaknesses. This choice of categorization seems quite interesting and meaningful since security vulnerabilities are often known by designers and are the natural causes of attacks on software systems. Alvi et al. proposed a vulnerability based scheme putting together security patterns and weaknesses documented in the CWE database [4]. They considered that the CWE weaknesses are appropriate to document flaws added through the design phase and they manually linked security patterns to CWE weaknesses from their textual descriptions. Anand et al. proposed another security pattern catalog composed of 12 families of vulnerabilities and identified some missing security patterns [6]. They focused on vulnerabilities because they considered that the CWE database is bigger than the scope of their work. They grouped software vulnerabilities into families and manually collected relationships between families and security patterns from the pattern textual descriptions and their vulnerability family definitions. We observed that these two classifications lack of navigability among patterns though, which is an important property defined as the ability to guide the choice of designers among related patterns [5]. More precisely, we noticed that some patterns for the same vulnerability family are compatible together (for example, Audit interceptor and Secure logger for log vulnerabilities [7]) and that others are conflicting in the sense that implementing conflicting patterns leads to inconsistencies in an application. As a consequence, a designer may be confused about the use of patterns.

As in [4], we propose a pattern classification expressing which patterns can be used to remove a weakness of the CWE database. But similarities end here. Indeed, our classification aims at proposing a more precise and accurate mapping between patterns and weaknesses. It is more accurate in the sense that we translate the meaning of the patterns and weaknesses into smaller properties, i.e. strong points and mitigations respectively. We establish relations among these properties with respect to security principles, which show the meaning of these relations. The principles are organized into a tree structure, which enables the generation of inter-dependences among the security patterns that overcome the same weakness. In addition, the classification is completed with the relationships among pattern themselves, expressing for example the conflicts or the dependence among patterns. This is why we claim that the proposed classification is more precise. Another strong contribution of this paper lies in the presentation of the methodology used to build this classification. This one is composed of seven manual and automatic steps, which offer the advantage to justify the pattern classification and reduces the efforts required to add new patterns or weaknesses to the classification.

III. Background

Our classification methodology is mainly based upon four security concepts: security patterns, weaknesses associated with their related mitigations, and security principles. We recall basics on these concepts below.

A. Security patterns

A security pattern is a generic solution to a recurrent security problem, which is characterized by a set of structural and behavioral properties [12]. It can be presented textually or with schema (UML diagrams). The quality of a pattern and its classification can be established with *strong points*, which correspond to sub-properties of the pattern [13]. Strong points are manually extracted from the forces and consequences of a security pattern.

In addition, a security pattern can be documented to express its relationships with other patterns. Yskout et al. proposed the following annotations between two patterns p_1 and p_2 [7]:

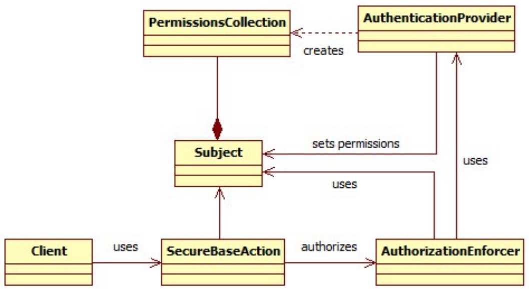


Figure. 1. Authorization enforcer pattern

- ”depend”means that the implementation of p_1 requires the implementation of p_2;
- ”benefit” expresses that implementing p_2 completes p_1 with extra security functionalities or decreases the development time. However, p_1 can be correctly implemented despite the absence of p_2;
- ”impair” means that the functioning of p_1 can be obstructed by the implementation of p_2;
- ”alternative” expresses that p_2 is a different pattern fulfilling the same functionality as p_1;
- ”conflict” encodes the fact that if both p_1 and p_2 are implemented together then it shall result in inconsistencies.

Such annotations may noticeably help combine patterns and not to devise unsound composite patterns.

For example, Figure 1 portrays the UML class diagram of the pattern ”Authorization enforcer” whose purpose is to check whether client applications are authorized to request a Web application. This security pattern structures an application in such a way that the authorization logic is centralized and decoupled from the functional logic of the application [14]. It benefits from the presence of a front controller (Security Base Action) offering a centralized entry point and from the presence of the pattern ”Authentication enforcer”, which encapsulates the authentication mechanism. It uses the ”Authentication Provider” whose role is to check the client credentials. With regards to its forces and consequences, the pattern ”Authorization enforcer” is characterized by the following strong points:

1) Minimize the decoupling between authorization and business logics;
2) URL based access control for Web applications;
3) Centralized authorization;
4) Systematic verification of client request permissions;
5) Promotes separation of responsibility;
6) Providing the application with authorization mechanism.

This pattern is related to three other patterns: it is an alternative to the pattern ”Container managed security”, and benefits from the patterns ”Secure service facade” and ”Authentication enforcer”.

B. CWE weaknesses and mitigations

The CWE database [15] provides an open catalog of software weaknesses. At the moment, this database includes around 700 software weaknesses but this number is still growing. A weakness is documented with a panoply of information, including a full description, its causes, detection methods, and a mapping between the weakness and CAPEC attack patterns or vulnerabilities. Furthermore, the CWE database provides, for each weakness, a set of potential mitigations. The latter summarize the actions to be done in order to overcome a weakness. A mitigation is identified by an id (MIT-ID) and is characterized by three important elements: a textual description, a strategy that corresponds to the basic security principle targeted by the mitigation, and its life cycle phase (requirements, design, implementation, etc.). Nevertheless, these elements are not supplied for all the mitigations of the database. In this work, we only consider mitigations that are fully documented for a better accuracy in the classification.

C. Security principles

A security principle is a desirable property, structure or behavior of software that aims at reducing the impact and the likelihood of a threat realization [16]. They represent an insight on the nature of close security tasks whose contexts are not taken into consideration. Saltzer and Schroeder firstly proposed a set of eight best practices for system security [17], which were widely expanded in the last decades to form security principles [16], [18]. Below, we present some security principles used throughout the paper:

1) Access control: This security principle collects the mechanisms allowing the identification/authentication of entities (users or services), the definition and the verification of their access rights and the accounting [19]. In particular, it includes the AAA (Authentication, Authorization and Accounting) principles.

2) In depth defense: Inspired from a military strategy, this security principle aims at protecting a system with a layered set of security mechanisms. It includes sub-principles and mechanisms that can be combined together in a layered form such as: ”Complete mediation” (input validation and canonicalization), ”Perimeter security” (e.g. firewalls), ”Intrusion detection and prevention”, ”Network and core trust zones definition” (e.g., demilitarized zones, sand boxes or chroot jails) [20].

3) Fault tolerance : From a safety point of view and in order to ensure an acceptable availability, Fault tolerance aims at enabling an application to continue operating normally (or in a reduced way) and enhancing the manageability of failures in the application or in some of its parts. Fault tolerance comprises the principle ”Exception management” to prevent the disclosure of internal information of the application in the case of a failure [18]. ”Repartition” is another sub-principle of Fault tolerance whose purpose is to avoid the whole application failure in the case of a dysfunction of one of its parts.

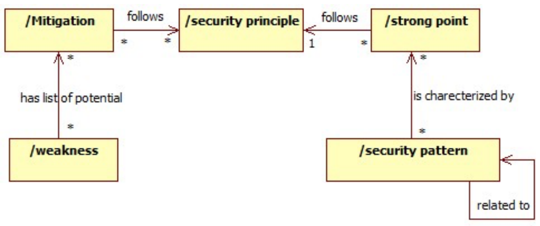


Figure. 2. The proposed mapping metamodel

4) Sensitive data : This principle addresses the prevention of sensitive data disclosures. For instance, it is constituted of these two well-known principles: ”Encryption” (referring to encryption methods of data) and ”Privacy” (referring to privacy protection) [18].

5) Configuration management: In software systems, it is advised to protect configuration items and to control item changes. The principle of ”Configuration management” expresses this need. It can also be implemented by other principles, which are related to configurable elements: ”Configuration protection”, ”Privilege management” or ”Fail-safe defaults” [18].

6) Security simplification: As software sophistication increases, so does the risk to add security vulnerabilities. This principle refers to the ”keep it simple stupid” (KISS) concept. It includes several sub-principles, e.g., the ”Economy of mechanisms”, ”Psychological acceptability” [17] or ”Open design” (avoid security by obscurity).

IV. THE CLASSIFICATION METHODOLOGY

Our methodology aims at inferring relationships among a security weakness, security principles and security patterns, expressing which set of patterns fixes the weakness and the relations among the patterns. Without loss of generality, we applied the following methodology on Web application weaknesses, but it can also be applied on other kinds of applications on condition that the dedicated weaknesses, mitigations and patterns are being well documented.

The mapping from a weakness to a set of security patterns is inferred with several steps, summarized in Figure 2. These steps aim at finding successive concrete links among security concepts. On the one hand, we look for the mitigations that avoid exploiting a given weakness. On the other hand, we anatomize security patterns to collect a set of strong points, which represent sub-properties of the pattern. Intuitively, we connect both sides by means of the notion of security principles because these give the objectives targeted by the mitigations and strong points that have to be joined.

The methodology outlined in Figure 2 is actually divided into 7 manual and automatic steps, presented in Figure 3. In the first step, we collect the security principles we are interested in and organize them into a hierarchy. Steps 2 to 5 establish several different mappings from weaknesses to mitigations, from mitigations to security principles, from security patterns to strong points and finally from strong points to security principles. In Step 6, these mapping are automatically inter-connected. We obtain a database Db_f from which the classification is automatically extracted in Step 7. Finally, for sake of readability, we depict pattern combinations with s, as illustrated in Figure 3 (right side). In our case, a SAG has a tree structure whose root node is labeled by a weakness. The other nodes are labeled by principles and the leaves by patterns. Relationships between two nodes can be defined using logic operations.

Our methodology offers the advantage to build a classification that can be easily updated. Indeed, if a new weakness is added to the CWE database, only the steps 2 and 3 have to be followed. In the same way, if a new security pattern is proposed in the literature, only the steps 4 and 5 have to be done. The re-generation of the whole classification, which includes new weaknesses or patterns is automatically performed. Each step is detailed below and illustrated with the weakness CWE-285: Improper Authorization.

A. Step 1: hierarchical organization of security principles

For this stage, we collected 66 security principles related to Web applications found in the literature and organized them into a tree, which reflects a hierarchy of principles from the most abstract (those nearest the top) to the most concrete ones (those nearest the bottom). Indeed, as presented in Section III, a security principle can be considered as the realization of other security principles, or as a subordinate principle of another one. This arrangement is represented by the tree depicted in Figure 4. This organization was manually established in relation to the nature of each security principle, often described with text. The proposed organization is not exhaustive but covers the security patterns and mitigations considered for this classification. We chose to organize security principles in this way to later generate inter-dependences among security patterns that completely or partially overcome a given weakness. In addition, this hierarchical organization shall give a complete overview (from the most abstract to the most concrete principles) on the nature of the security mechanisms that are required to remove a weakness and, in the same time, that are provided by security patterns.

We encoded this information in a database denoted DB_1.

B. Step 2: weakness and mitigation extraction

In this step, we automatically extracted the weaknesses related to Web applications from the CWE database, and for each weakness, its potential mitigations. The result is stored in a database denoted DB_2, which gathers information about 185 CWE weaknesses and 65 mitigations.

For example, the weakness CWE-285: Improper Authorization is commonly found in Web applications and occurs when an application does not correctly manage authorization checks. When the application does not have a correct authorization mechanism, it exposes several vulnerabilities, e.g., denial of service or arbitrary code execution. This weakness can be fixed by means of several mitigations. Due to lack of room, we only expose some of them here:

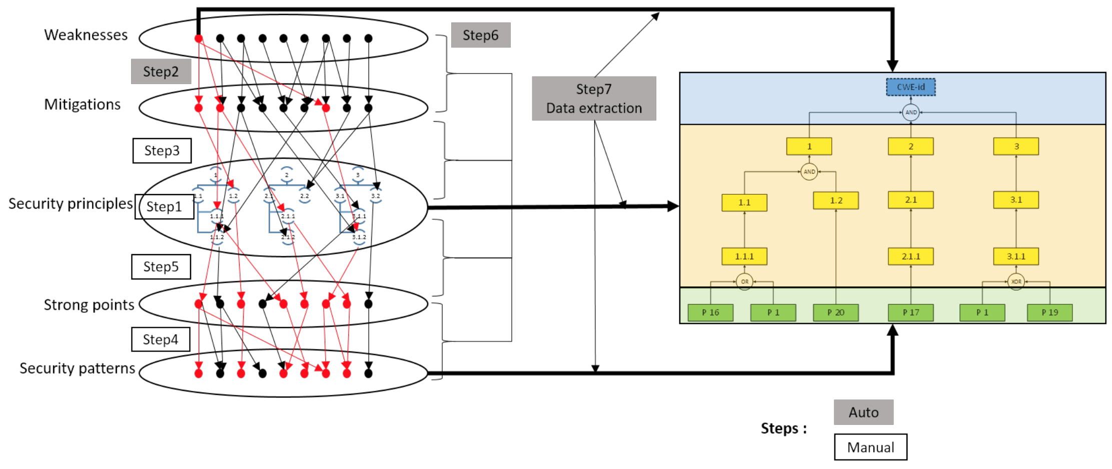


Figure. 3. Classification methodology

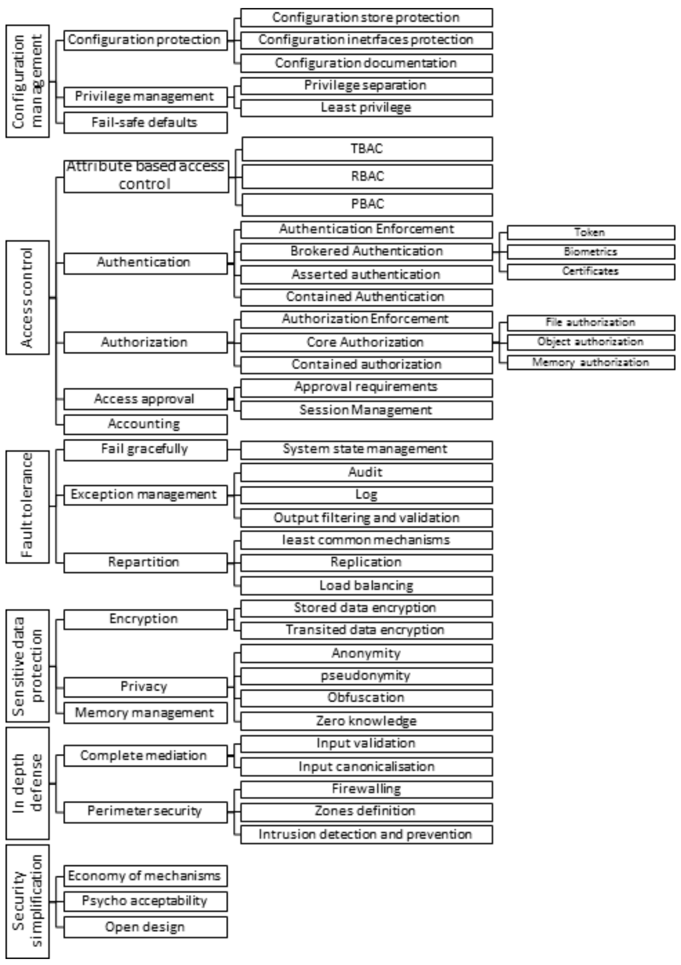


Figure. 4. Proposed core and design security principles

- Use of a framework that correctly performs the authorization checks;
- Divide the software into anonymous, normal, privileged, and administrative areas;
- Use Role Based Access Control;
- Default deny in access control lists (ACLs);
- Authorization correctly enforced at the server side;
- Restrict access to requests having an active authenticated session.

C. Step 3: mapping between mitigations and security principles

This step aims at establishing relations between mitigations and security principles. It was manually done by interpreting the strategies found in mitigations. Indeed, the meaning of a mitigation strategy is often very close or comprised into some security principle definitions. As a strategy may cover several security principles, we have to consider a many-to-many relation here.

Mitigations are usually described with concrete mechanisms given with a low abstraction level. Hence, we observed that mitigations are often associated with the most concrete security principles in reference to the hierarchical organization of Figure 4. This step gives a database DB_3 associating 65 mitigations and 66 security principles.

For example, the weakness CWE-285 can be fixed with the mitigation Divide the software into anonymous, normal, privileged, and administrative areas, which is related to two security principles:

- Least common mechanisms: so that a failure in an area does not affect another area;
- Privilege separation: so that a privilege given in an area is not valid in the other area. A spoofed identity does not give the possibility to compromise all the authorization in the application.

D. Step 4: mapping between patterns and strong points

We established two relations among patterns and strong points:

1) the first one is a many-to-many relation between security patterns and strong points, each pattern being characterized by a set of strong points that can be shared with other patterns. For example, the patterns "Authorization enforcer" and "Container managed security" share the strong point "Providing the application with authorization mechanism". This relation is established by manually extracting the strong points of each pattern from its textual description (forces and consequences of the pattern);
2) the second relation is related to the annotations "depend", "benefit", "impair" or "alternative" defined among patterns [7]. With P a set of patterns, this relation is defined as a mapping from P^2 to the annotation set $\{"depend", "benefit", "impair", "alternative"\}$, which provides for every pair of patterns $(p1, p2)$ an annotation about the relationship between $p1$ and $p2$.

To summarize, these relations provide connections among patterns and between patterns and strong points. These are encoded into the database DB_4, associating 26 security patterns and 36 strong points.

E. Step 5: mapping between strong points and security principles

As introduced in [21], security patterns are classifiable w.r.t. security principles, like most of the security techniques. Instead of directly looking for a relation between patterns and security principles, we focus on the strong points, which are more precise sub-properties themselves satisfied in patterns. Hence, this step aims at establishing a many-to-many relation between strong points and principles. This step was manually done since strong points and principles are mostly presented with textual documents. During this step, we observed that the abstraction level of strong points better fit with the most concrete principles exposed in our hierarchical organization of Figure 4. This relation is materialized with the database DB_5.

For example, the strong point "Minimize the decoupling between authorization and business logics" of the pattern "Authorization Enforcer" has a security objective also found in the principle "Economy of mechanism", which is a sub-principle of "Security simplification".

F. Step 6: data consolidation

This automatic step integrates the databases DB_3, DB_4, and DB_5 into a single one. On the one hand, DB_3 stores the relations among CWE weaknesses, their potential mitigations and the related security principles. On the other hand, DB_4 and DB_5 store the relations among security patterns, strong points and principles. It is manifest that the security principle hierarchy becomes the central point that interconnects weaknesses with security patterns. We automatically performed this step with the tool Talend [1], which produces the final database DB_f.

[1]https://www.talend.com

G. Step 7: classification extraction

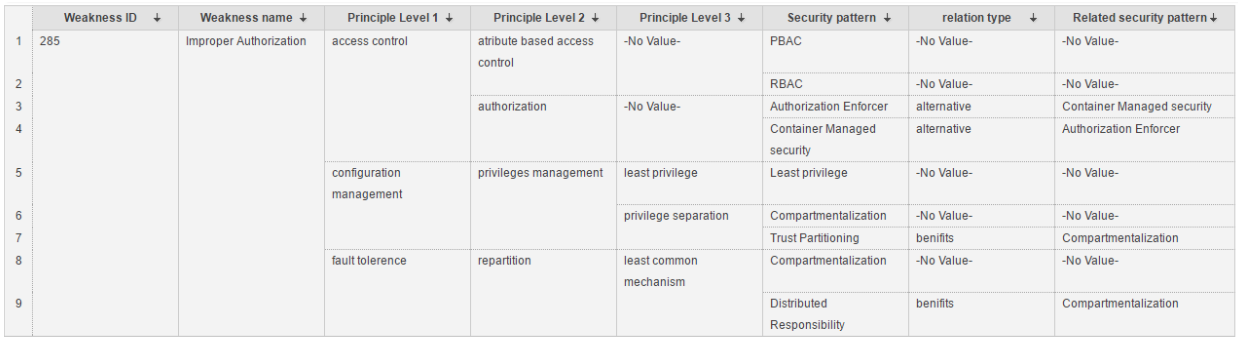
The database DB_f holds all the information required to extract a pattern classification. We have chosen to catalog the combinations of patterns that prevent attackers from exploiting a given weakness. This step automatically collects these combinations of patterns for every weakness found in DB_f. More precisely, for a given weakness, we extracted:

- the information about the weakness (name, identifier, description, etc.);
- the complete hierarchy of security principles Sp related to a weakness, i.e. the arrangements of principles from the most abstract ones to the most concrete principles. The principles of Sp are associated with the weakness according to its potential mitigations given by the CWE database. The latter does not precise if one of the proposed mitigations is sufficient to fix the weakness or if all the mitigations have to be applied. As a consequence, we suppose that all the security principles have to be applied in order to overcome the weakness;
- for every principle sp in Sp, the set of patterns P_{sp}, the set of patterns $P2_{sp}$ not in P_{sp} that have relations with any pattern of P_{sp}, and the nature of these relations defined for couples of patterns by the annotations in $\{"depend", "benefit", "impair", "alternative", "conflict"\}$.

Figure 5 shows an example of data extraction achieved for the weakness CWE-285. The tabular provides the security principles and their levels in the tree of Figure 4 (col. 3-5), the related security patterns (col. 6) and the relations among patterns (col. 7,8). Strong points and mitigations can also be extracted.

For sake of readability, we propose to portray this classification with SAGs, organizing the security principles and patterns related to a weakness. SAGs are semi-automatically drawn by these steps:

1) a weakness has its own SAG whose root is labeled by the weakness identifier. The root node is linked to the most abstract principles found in Sp connected together with the AND operator, since we consider that all the security principles have to be ensured to remove the weakness. The security principles are themselves connected, from the most to the less abstract ones, by keeping the hierarchical organization defined in Step 1;
2) as our classification provides the relations among patterns (as explained in Step 4), we propose, in this final step, to complete the SAG with new nodes encoding these relations. Actually, we replace these relations with logic operators to infer a Boolean expression from all these relations. Given a security principle sp in Sp and a couple of patterns (p_1, p_2) of the set P_{sp}, if we have:
   - $(p_1\ depend\ p_2)$ or $(p_1\ benefit\ p_2)$, we use the expression $(p_1\ AND\ p_2)$;
   - $(p_1\ impair\ p_2)$, we use $(p_1\ XOR\ p_2)$ since the presence of p_2 can decrease the efficiency of p_1;



	Weakness ID	Weakness name	Principle Level 1	Principle Level 2	Principle Level 3	Security pattern	relation type	Related security pattern
1	285	Improper Authorization	access control	atribute based access control	-No Value-	PBAC	-No Value-	-No Value-
2						RBAC	-No Value-	-No Value-
3				authorization	-No Value-	Authorization Enforcer	alternative	Container Managed security
4						Container Managed security	alternative	Authorization Enforcer
5			configuration management	privileges management	least privilege	Least privilege	-No Value-	-No Value-
6					privilege separation	Compartmentalization	-No Value-	-No Value-
7						Trust Partitioning	benifits	Compartmentalization
8			fault tolerence	repartition	least common mechanism	Compartmentalization	-No Value-	-No Value-
9						Distributed Responsibility	benifits	Compartmentalization

Figure. 5. Extraction of the pattern classification for the weakness CWE-285

- $(p_1\ alternative\ p_2)$, we use $(p_1\ OR\ p_2)$. The use of these two patterns together increases the complexity of the system, but is not problematic. For example, the patterns Authorization Enforcer and Container Managed Security are alternative;
- $(p_1\ conflict\ p_2)$, we use the expression $(p_1\ XOR\ p_2)$ meaning that only one of the pattern can be used;
- p_1 having no relation with any pattern p_2 in P_{sp}, we add the expression p_1.

All these expressions are assembled with the "AND" operation. The resulting Boolean expression is graphically shaped with an expression tree whose nodes are logic operations and leaves are patterns. The expression tree is linked to the security principle sp.

3) we observed that the number of relations among patterns may be large and not always relevant. As a consequence, a designer may still be confused about the choice of the patterns to use, especially when there are conflicted patterns. Hence, we propose to simplify Boolean expressions and to update SAGs with simplified expression trees. The Boolean expression reduction is here performed with the tool BExpRed [2]. For instance, with the three patterns p_1, p_2 and p_3 having the relations $(p_1\ benefit\ p_2)$, $(p_1\ alternative\ p_3)$ and $(p_2\ alternative\ p_3)$, we obtain $(p_1\ AND\ p_2)$ $AND\ (p_2\ OR\ p_3)\ AND\ (p_1\ OR\ p_3)$, which can be simplified into the expression $(p_1\ AND\ p_2)$. It is manifest that this expression is clearer than the first one.

The final SAG depicts the combinations of patterns, which overcome a weakness. We believe that these SAGs offer a good point of view on the potential solutions and can help choose the most appropriate one with regard to the application context. Figure 6 depicts the final SAG obtained for the weakness CWE-285. It shows that designers can implement either "Authorization Enforcer" or "Container managed security" and should implement all the other patterns (PBAC, RBAC, Least privileges, etc.) in order to fix the weakness. The SAG also shows the security principles applied here.

[2]https://sourceforge.net/projects/bexpred/

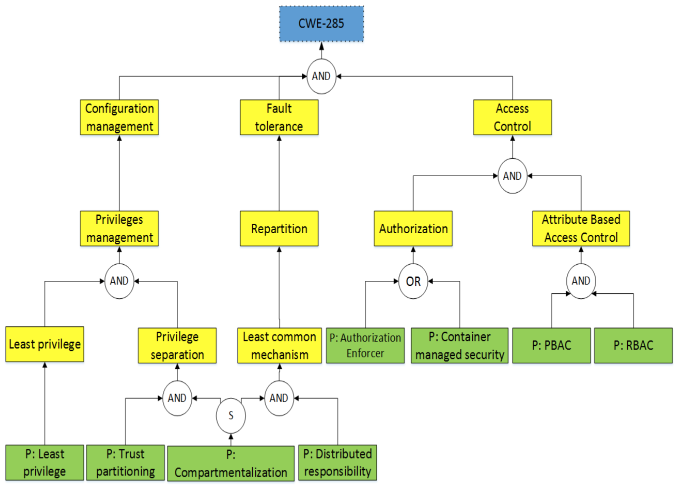


Figure. 6. CWE-285 Security patterns tree

V. Classification Short Presentation

The classification is built on 185 weaknesses, 26 security patterns and 66 principles. Due to lack of room, the complete list is available, with the classification, in [10]. Presented in tabular form, as illustrated in Figure 5, it enables multi-attribute based decisions insofar as patterns can also be classified according to shared strong points, supported security principles, related weaknesses, mitigations and inter-patterns relationships.

The proposed classification complies with several quality criteria defined in [5]. Among them, we have noted Navigability, Unambiguity and Usefulness classification. Navigability, which is defined as the ability to direct designers among related patterns, is satisfied since the classification provides relationships among patterns and SAGs that illustrate the different choices of pattern combinations. Unambiguity is taken into account since the classification is clearly defined by means of the methodology steps, which provide relations among security properties. All these steps justify the classification. We believe that the classification can be used in practice since it is based upon the CWE database (weaknesses, mitigations) and security principles. In addition, SAGs can help designers in their pattern combination choices without ambiguity.

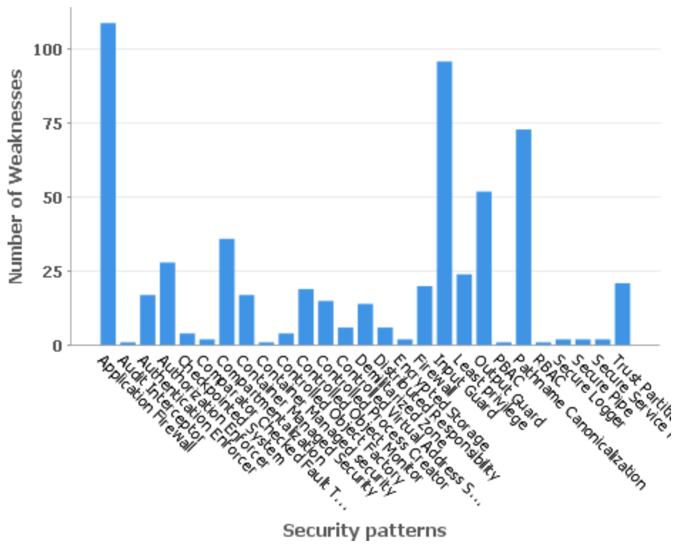


Figure. 7. Number of fixed weaknesses per pattern

A variety of statistical information can be extracted from the classification. For instance, Figure 7 shows the number of partly fixed weaknesses per pattern. Keeping in mind, that the set of patterns taken into consideration is not exhaustive, 4 patterns seems to emerge for partly fixing a large part of the 185 weaknesses covered by the classification: ”Input Guard”, ”Output Guard”, ”Pathname Canonicalization” and in particular ”Application firewall”, which can overcome 109 weaknesses. This kind of information shows that, completed with more patterns, our classification and methodology would guide designers on good practices. For instance, in our context, one can deduce that these 4 patterns have to be used with Web applications. Unfortunately, the classification also reveals that 26 security patterns is insufficient to fix all the Web application weaknesses. Indeed, 30 weaknesses are not yet associated with security patterns. Hence the need to complete the current classification.

VI. Conclusion

In this paper, we presented a classification methodology putting together software weaknesses, security principles and patterns in order to help designers in the choice of the best pattern combination to fix a given weakness. This methodology is composed of some manual steps subdividing weaknesses and patterns into detailed security properties. Then, these are automatically associated together with respect to security principles. We organized the latter into a hierarchy to precisely express the security objectives of these properties. The resulting classification is stored into a database available in [10]. This database can be upgraded with new patterns and weaknesses without the need of re-writing the whole classification. We also proposed to portray this classification by means of Security Activity Graphs showing, without ambiguity, combinations of patterns and their relations.

In future research, we will continue investigating the specific issues raised by security pattern classification. Indeed, their heterogeneous nature and sometimes their ambiguous descriptions leave several problems open. For instance, we intend to investigate whether text mining techniques could help to partially automate some steps of the methodology (steps 2 to 5). We also intend to complete the classification with the attack patterns provided by CAPEC (Common Attack Patterns Enumeration and Classification) to provide another point of view on the pattern use. Indeed, it should be possible to extract Attack Defence Trees (ADT) [22] depicting the defences in terms of security patterns that designers may use to protect a software system against attacks.

References

[1] J. Yoder, J. Yoder, J. Barcalow, and J. Barcalow, “Architectural patterns for enabling application security,” *Proceedings of PLoP 1997*, vol. 51, p. 31, 1998.

[2] Security patterns repository. [Online]. Available: http://sefm.cs.utsa.edu/repository/

[3] K. Yskout, R. Scandariato, and W. Joosen, “Does organizing security patterns focus architectural choices?” *Proceedings - International Conference on Software Engineering*, pp. 617–627, 2012.

[4] A. K. Alvi and M. Zulkernine, “A Natural Classification Scheme for Software Security Patterns,” *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 113–120, 2011.

[5] K. Alvi, Aleem and M. Zulkernine, “A Comparative Study of Software Security Pattern Classifications,” *2012 Seventh International Conference on Availability, Reliability and Security*, pp. 582–589, 2012.

[6] P. Anand, J. Ryoo, and R. Kazman, “Vulnerability-Based Security Pattern Categorization in Search of Missing Patterns,” *2014 Ninth International Conference on Availability, Reliability and Security*, pp. 476–483, 2014.

[7] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen, “A system of security patterns,” 2006.

[8] E. B. Fernandez, H. Washizaki, N. Yoshioka, A. Kubo, and Y. Fukazawa, “Classifying security patterns,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4976 LNCS, 2008, pp. 342–347.

[9] S. Ardi, D. Byers, and N. Shahmehri, “Towards a structured unified process for software security,” in *Proceedings of the 2006 international workshop on Software engineering for secure systems*. ACM, 2006, pp. 3–10.

[10] Security pattern classification. [Online]. Available: http://regainia.com/research/database.html

[11] M. Bunke, R. Koschke, and K. Sohr, “Organizing security patterns related to security and pattern recognition requirements,” *International Journal on Advances in Security*, vol. 5, 2012.

[12] E. B. Fernandez, “Security patterns and secure systems design,” pp. 233–234, 2007.

[13] D. Harb, C. Bouhours, and H. Leblanc, *Using an Ontology to Suggest Software Design Patterns Integration*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 318–331. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01648-6_34

[14] C. Steel, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall PTR, 2005.

[15] Common weakness enumeration.

[16] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way, Portable Documents*. Pearson Education, 2001.

[17] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[18] J. Meier, “Web application security engineering,” *Security & Privacy, IEEE*, vol. 4, no. 4, pp. 16–24, 2006.

[19] J. Scambray and E. Olson, *Improving Web Application Security*, 2003.

[20] M. R. Stytz, “Considering Defense in Depth for Software Applications,” *IEEE Security and Privacy*, vol. 2, no. 1, pp. 72–75, 2004.

[21] R. Wassermann and B. H. Cheng, “Security patterns,” in *Michigan State University, PLoP Conf.* Citeseer, 2003.

[22] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, “Attack–defense trees,” *Journal of Logic and Computation*, p. exs029, 2012.