A security pattern classification based on Data integration

Sébastien Salva, Loukmen Regainia

A security pattern classification based on Data integration

Sébastien Salva[1] and Loukmen Regainia[2]

[1] LIMOS CNRS UMR 6158, Clermont Auvergne University,
sebastien.salva@uca.fr
[2] LIMOS CNRS UMR 6158, Clermont Auvergne University,
loukmen.regainia@uca.fr

Abstract. Security patterns are design patterns specialised to provide reusable and general solutions to recurring security problems. These patterns, which capture the strengths of different security approaches, are intended to make the design of maintainable and secure applications easier. The pattern community is continuously providing new security patterns (180 patterns are available at the moment). For a given problem, this growing pattern set along with their abstract presentations make the security pattern choice tedious, even for experts in software design. We contribute in this issue by presenting a method of *security pattern classification* based upon data extraction and integration. The pattern classification is semi-automatically inferred by means of a data-store integrating disparate publicly available security data. This classification exposes relationships among software attacks, weaknesses, security principles and security patterns. It expresses the pattern combinations that can counter a given attack. Besides the pattern classification, we show that the data-store can be used to generate *Attack Defense Trees*. In our context, these illustrate, for a given attack, its sub-attacks and the related defenses given under the form of security pattern combinations. Such trees make the pattern classification more readable even for beginners in security patterns. Finally, we evaluate on 25 human subjects the benefits of using Attack Defense Trees and a classification established for Web applications, which covers 215 attacks, 136 software weaknesses, 66 security principles and 26 security patterns.

Keywords: Security patterns, Classification, Data integration, CAPEC attacks, CWE weaknesses, Attack-Defense Trees

1 Introduction

Design patterns are recurring solutions to software design problems proposed and used by skilled application or system designers. They are more and more considered in the industry since they may accelerate the design stage of the software life cycle and help in the code readability and maintenance. As the interest in software security continuously grows for a few years, specialised patterns also emerged to help design secure applications. These, called security patterns,

are defined as *reusable elements to design secure applications, which will enable software architects and designers to produce a system that meets their security requirements and that is maintainable and extensible from the smallest to the largest systems* [19]. Schumacher also postulates that *Security patterns relates countermeasures to threats and attacks in a given context* [21].

Security patterns are often presented at a high level of abstraction with texts and sometimes with UML diagrams to be reusable in different kinds of context. Since 1997, the number of security patterns is continuously growing. The repository given in [22] lists around 180 security patterns. Because of the abstract nature of security patterns and because the documents are not structured in the same manner, the choice of the most appropriate pattern to solve a security problem is difficult with regard to a given context and somehow perilous for novice designers [1] as a wrong choice may imply the use of useless countermeasures or the addition of new vulnerabilities in the design and code of the application. As designers cannot be experts in all the software engineering fields, security pattern classifications have been published in the literature to help them find the most appropriate patterns according to the application requirements. Several classifications were proposed to arrange security patterns into different categories, e.g., by security principles [28, 2], by application domains [4] (software, network, user, etc.), by vulnerabilities [3, 1] or by attacks [27, 1]. Despite the improvements in the pattern choice, several issues still remain open. Among them, we noticed that these classifications are manually devised by directly comparing the textual descriptions of security concepts (vulnerabilities, attacks, patterns, etc.). As these descriptions are generic and have different purposes, the categorisation of a pattern can be done only when there is an evident relation between it and other security concepts. Besides, as these classifications are not *deterministic* (no strict definition of the classification process [2]), it becomes often delicate to upgrade them. These observations lead to the purpose of this paper, which is to establish a strict security pattern classification method, composed of several successive steps. These lead to a pattern classification that organises the security patterns that can be used to counter an attack. This paper extends the work we initiated in [16] and includes the following contributions:

- we present a data-store architecture and an integration method that extracts data from various Web and publicly accessible sources and stores relationships among attacks, security principles and security patterns. The data-store integrates data coming from the CAPEC base [12], several papers dealing with security principles [20, 25, 11, 5, 10] and from the pattern catalogue given in [29]. It also integrates the studies about inter-pattern relations [28, 6]. All these steps provide the detailed justifications of the resulting classification;
- we automatically derive a security pattern classification from the data-store, providing the pattern combinations that can be used to counter an attack. The inter-pattern relations integrated into the data-store offer the advantage of making apparent the dependencies among patterns as well as the conflicting or alternative patterns. We have generated a classification spe-

cialised to Web applications, which includes 215 CAPEC attacks, 136 CWE weaknesses, and 26 security patterns covering varied security aspects. To the best of our knowledge, this is the largest classification for this application domain. It is stored in a database available in [17];
- we automatically generate Attack-Defence Trees (shortened ADTrees [9]), which aim at supplementing the classification with illustrations depicting, for a given attack, its sub-attacks along with defenses expressed here with security patterns. Such ADTrees improve the understanding of the previous classification.

We employed the classification to evaluate, on 25 human subjects, the benefits of using our pattern classification and ADTrees in terms of Comprehensibility, Effectiveness and Accuracy.

Paper organisation: Section 2 presents the related work and the motivations of the approach. In addition, we introduce some security notions and data used throughout the paper. The extraction and integration of security data into the data-store are given in Section 3. The next section shows how we automatically extract the pattern classification and ADTrees from the data-store. Then, we evaluate it in Section 6. We discuss about the advantages and limitations of our classification in Section 7. We traditionally conclude and give some perspectives for future work in Section 8.

2 Background

2.1 Related work and motivations

The growing number of security patterns available in the literature makes the choice of the most appropriate ones very difficult for overcoming a security problem. In order to ease this task, several pattern catalogues and classifications were proposed [22, 14, 29, 27, 23, 1, 24, 3]. An overview of these documents is given by Bunke et al., who reviewed the papers dealing with security patterns between 1997 and 2012 [4]. They listed a set of classification criteria and established a comparison between design patterns and security pattern classifications. They finally proposed their own classification based upon the application domains of patterns (software, network, user, etc.)

Vulnerabilities are taken into consideration for pattern classification in [3, 1]. Compared to the previous paper, these give another point of view helping designers in the choice of patterns to fix software vulnerabilities. The classifications exposed in [27, 23, 1, 24] expose pattern categories by focusing on the attacker side and attacks. This choice of categorisation seems quite interesting and meaningful as attacks are more and more known and examined by designers. Wiesauer et al. initially introduced in [27] a short taxonomy of security design patterns made from links found in the textual descriptions of attacks and security patterns. Wiesauer et al. claimed that 40 security patterns can be connected to attacks, but only few examples are given. These examples are associated with one or two patterns only. Tondel et al. presented in [23] the combination of

three formalisms of security modelling (misuse cases, attack trees and security activity models) in order to give a more complete security modelling approach. In their method for building attack trees, they linked some activities of attack trees with CAPEC attacks; they also connected some activities of SAGs (security activity diagrams) with security patterns. The relationships among security activities and security patterns are manually extracted from documentation and are not explained. Shortly after, Alvi et al. presented a classification scheme for security patterns putting together CAPEC attacks and security patterns for the implementation phase of the software life cycle [1]. They analysed some security pattern templates available in the literature and proposed a new template associated with software lifecycle phases. They considered around 20 attacks and linked them to 5 patterns. They also manually augmented the CAPEC attack documentation with a section named "*Relevant security patterns*" composed of some patterns [1]. After inspecting the CAPEC base, we observed that this section is seldom available, which limits its use and interest. Finally, Uzunov et al. introduced in [24] a classification of security threats and patterns specialised for distributed systems. They proposed a library of threats and their relationships with security patterns in order to reduce the expertise level required for the design of secure applications. They considered that that their threat patterns are abstract enough to encompass security problems related to the context of distributed systems [24].

Open issues and contributions

Alvi et al. outlined 24 pattern catalogues and classifications in [2] and established a comparative study to point out their positive and negative aspects. They chose 29 classification attributes (purpose, abstraction levels, life-cycle, etc.) and compared the classifications against a set of nine 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. Yskout et al. also reported that the security pattern adoption is limited *possibly due to a sub-optimal quality of the documentation* [29]. We indeed believe that security pattern classifications lack Navigability and Comprehensibility, which are quality criteria respectively defined as: the ability to direct a software designer among collaborative and related patterns; the ease to understand patterns by both a novice and expert developer.

We also observed that the main issue of the above works lies in the lack of a precise method to build the classification. All of them are based upon the interpretation of different documents, which are converted to abstract relationships. The first consequence of these interpretations is the difficulty to extend these classifications. In addition, it is sometimes tricky to understand the reasons of the relationships established between attacks and patterns.

In [18], we introduced a first semi-automatic classification method and the classification itself, which exposes relationships among 185 software weaknesses

of the CWE base [13], security principles and 26 security patterns. The classification groups the patterns that partially mitigate a given weakness with respect to the security principles that have to be addressed to fix the weakness. In [16], we introduced another classification method to categorise the security patterns that can be used to counter attacks. We extend this work by describing the meta-model of the data-store used to automatically infer the pattern classification. The classification process, which is built on data acquisition, is composed of six manual and automatic steps. They offer the advantage of justifying the pattern classification and reduce the efforts required to add new patterns or attacks to the classification. Finally, we complete the classification with ADTrees illustrating attacks, sub-attacks and security patterns as defenses. These are generated after the choice of an attack in the classification and remain up-to-date.

2.2 Publicly accessible resources used for the data integration

Security patterns A security pattern is a generic solution to a recurring security problem, which is characterised by a set of structural and behavioural properties. A pattern is described with textual sections called intents, forces and consequences. These sections point out the features of a pattern, called *strong points* [7]. For a security pattern, strong points characterise the forces and the consequences brought by the use of the pattern against a security problem.

In addition, a security pattern can be documented to express its relationships with other patterns. Such annotations may noticeably help combine patterns and not to devise unsound composite patterns. Yskout et al. defined the following annotations between two patterns p_1 and p_2 [28]:

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

For example, Figure 1 portrays the UML class diagram of the pattern "Application Firewall" whose purpose is to filter requests and responses to and from an application, based on access control policies. This security pattern structures an application in such a way that the inputs filtering logic is centralised and decoupled from the functional logic of the application. This is a strong point of this pattern. "Application Firewall" is related to two other security patterns [28]: it is an alternative to the patterns "Input Guard" and "Output Guard" since it is able to filter input calls and output responses from the application.

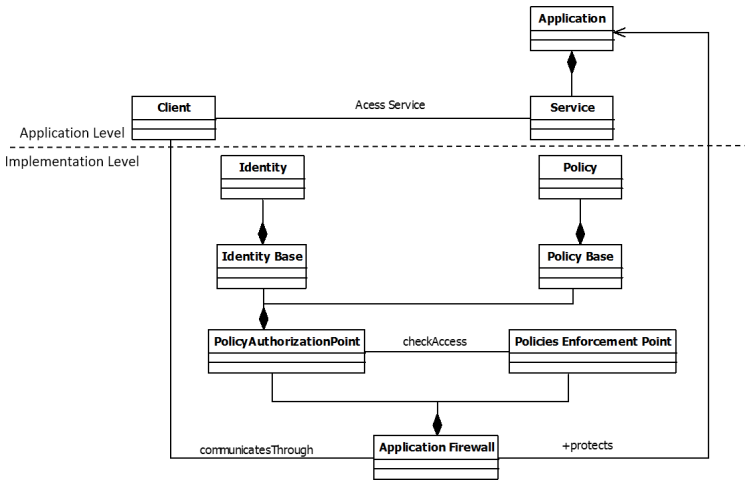


Fig. 1. Security pattern "Application Firewall", reprinted from Security Pattern Catalog, URL: https://people.cs.kuleuven.be/ koen.yskout/icse15/catalog.pdf, 2017

CWE weaknesses The Common Weakness (CWE) base [13] provides an open catalogue of software weaknesses, which are software or design mistakes that can lead to vulnerabilities. At the moment, this database includes around 1000 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 relations with CAPEC attacks or vulnerabilities. In addition, a set of potential mitigations are often proposed.

Capec Attacks The Common Attack Pattern Enumeration and Classification (CAPEC) is an open database offering a catalogue of attacks in a comprehensive schema [12]. Attack patterns are descriptions of common approaches that attackers take to target weaknesses of software or systems. An attack pattern, which we refer here as documentation (to avoid the confusion with security pattern), consists of several textual sections, e.g., *Attack Execution Flow, Severity*, etc. In our context, three sections sound particularly interesting for starting a classification. The section *Related attack patterns* shows interdependence among attacks, having different levels of abstraction. The first two levels (Category and Meta pattern) give attack mechanisms, the last two levels called "Standard pattern" and "Detailed attack pattern" gather the most concrete attacks. These interdependences provide a hierarchical organisation of attacks. Another section called *Related Weaknesses* lists the CWE weaknesses targeted by the attack. The section "Relation security principles" aligns some principles defined as desirable properties targeted by the attacks. At the moment, this section is often incomplete though.

Security principles A security principle is a desirable property, structure or behaviour of software that aims at reducing the impact and the likelihood of a threat realisation [25]. They represent an insight on the nature of close security tasks whose contexts are not taken into consideration.

Numerous works focused on security principles since the last four decades. Saltzer and Schroeder firstly established a set of eight best practices for system security [20], which were widely expanded to form security principles [25, 11, 5, 10]. Most of these papers reflect the fact that a security principle has a level of abstraction; it may be the realisation of other security principles, or as a subordinate principle of another one.

3 Data-store architectures

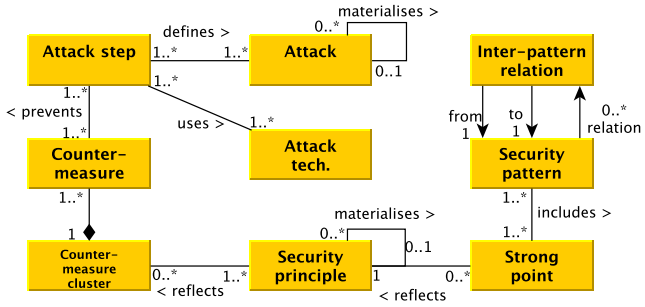


Fig. 2. Metamodel 1 of the data-store

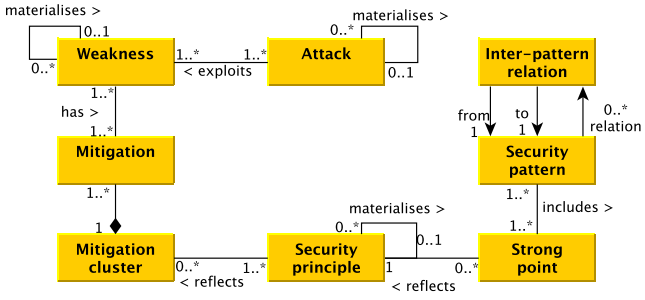


Fig. 3. Metamodel 2 of the data-store

As stated previously, our classification aims to make the design of secure applications easier by providing the set of security patterns that can be used as countermeasures against a given attack (in reference to the security pattern definition of Schumacher [21]) and the relations among these patterns. Finding

direct relations among attacks and security patterns by reading documentation is a hard problem. The documents are presented quite differently, with different level of abstractions. Instead, in order to later infer a precise classification, we chose to anatomise the security concepts available in documentation into more detailed properties that can be interconnected in an explicit manner. The literature and some attack bases [12, 13, 15] have confirmed to us the importance of the following associations: an attack can be documented with more concrete attacks, which can be themselves segmented into steps; These steps can be performed with techniques and can be prevented with countermeasures. These properties and associations are modelled with the meta-model of Figure 2. Besides, an attack also exploits a weakness, which may be composed of several more concrete weaknesses. Mitigations can be applied to treat them. These others associations are illustrated in Figure 3. As for security patterns, they can be characterised with strong points, which are pattern features that are extractable from pattern descriptions. In addition, a security pattern can have relations with others patterns. Figures 2 and 3 depict these properties and relations with entities in the same way.

Countermeasures, mitigations and strong points refer to the notion of attack prevention. But directly finding relations among them is still an obscure task as these properties have initially different purposes. To solve this issue, we chose to focus on security principles as mediators. As introduced by Wassermann et al., security patterns are classifiable w.r.t. security principles like most of the security concepts [26]. Here again, we consider that a security principle are organised into a hierarchy, which shows the materialisation of a principle with more concrete ones. Countermeasures and mitigations are often detailed security properties. It turns out that gathering them into groups (clusters) often reduces the efforts required to find connections with security principles without adding ambiguity. The choice of the cluster granularity, i.e., the size of the groups, along with the principle organisation offer a lot of flexibility to reach about the same abstraction level among strong points, principles, countermeasures and mitigations. In other words, these techniques help associate clusters, principles and strong points. These last security properties and associations are identically modelled in Figures 2 and 3.

Both meta-models of Figures 2 and 3 can be used to structure our datastore. A last possible meta-model could be achieved by blending the two previous ones. At the moment, we prefer avoiding this solution as the countermeasures of an attack step and the mitigations of a weakness have different purposes. We believe that gathering them might bring confusing associations among security principles and clusters, and finally false relations among attacks and security patterns. After inspecting the available security data resources, e.g., [12, 13, 15], we observed that few documents provide the countermeasures of a given attack step. For instance, some countermeasures are provided in the CAPEC base, but not all of them. In contrast, many countermeasures are listed for a weakness in the CWE base. In essence, it is manifest that the more security data we collect, the more precise the pattern classification will be. This is why we prefer using

the meta-model of Figure 3 for designing our data-store. The next section shows how the data integration is performed with this data-store.

4 Data integration

We present, in this section, the six steps required to integrate security data into the data-store. These aim at collecting security data and establishing the different relations illustrated in the meta-model of Figure 3. Steps 1 to 5 give birth to databases, and Step 6 consolidates them so that every entity of the meta-model is related to the other ones as expected. The steps 1,2 and 6 are automatically done with tools. These steps offer the strong advantage of semi-automatically achieving a data-store, which can be updated. For instance, if one wants to add a new attack, the steps 1 and 2 have to be followed. Likewise, if a new security pattern is available in the literature, the steps 3 and 5 have to be applied.

We have implemented these steps with scripts mostly based upon the tool Talend[3], an ELT (Extraction, Load, Transform) tool that allows an automated processing of data independently from the type of its source or destination. We applied these steps on attacks, patterns and principles related to the Web application context and on data coming from different sources: the CAPEC and CWE bases, several papers dealing with security principles [20, 25, 11, 5, 10] and the pattern catalogue given in [29]. We provide some quantitative results related to this context with each step. But other kinds of systems can be considered as long as documentation is available.

We also illustrate these steps with the pattern "Application Firewall" and with the attack "CAPEC-39: Manipulating Opaque Client-based Data Tokens", which corresponds to a threat on applications using tokens, e.g., cookies, holding personal data.

4.1 Step 1: CAPEC attack extraction and organisation

We chose to focus on the CAPEC base to extract information about security attacks because this appears to be the most complete base composed of the largest number of attacks explained in detail (steps, techniques, risks, security controls, etc.)

We extracted attacks from the CAPEC base and organised them into a single tree that describes a hierarchy of attacks from the most abstract to the most concrete ones so that we can get all the sub-attacks of a given attack. To reach that purpose, we rely on the relationships among attack descriptions found in the CAPEC section called *Related Attack Patterns*. By scrutinising all the CAPEC documents, it becomes possible to develop a hierarchical tree whose root node is unlabelled and connected to the most abstract attacks of the type "Category". These nodes are parents of attacks that belong to the type "Meta Attack pattern"

[3] https://talend.com/

and so on. The leaves are the most concrete attacks of the type "Detailed attack pattern".

The relations among attacks ("parent of", "child of") are provided in the CAPEC Base. Figure 4 shows the related attacks for the attack CAPEC-39. The abstraction level of the attack is expressed in the column "Type" (M stands for Meta-pattern, C for Category, D for Detailed pattern), the links with other attacks are listed in the column "Nature". Figure 4 shows the CAPEC-39 has one sub-attack "CAPEC-31: Accessing/Intercepting/Modifying HTTP Cookies".

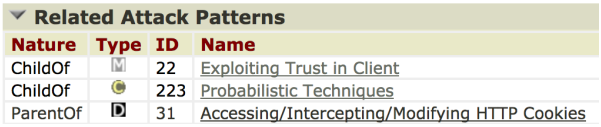


Fig. 4. Hierarchical organisation of attacks for the attack CAPEC 39, adapted from the CAPEC base, URL:https://capec.mitre.org/, 2017

This data extraction is automatically performed with a script, which yields a database DB_1. From the CAPEC database Version 2.8, we collected 215 attacks for the Web application context.

4.2 Step 2: CWE weakness and mitigation extraction

Given an attack of the database DB_1, we automatically extracted the CWE weaknesses targeted by the attack. These can be found in a textual section called *Related Weaknesses* of the CAPEC documents. Weaknesses are grouped here into two categories named Targeted and Secondary ranking the impact degree of the attack on a weakness. We focused on the type Targeted (even though it could also be relevant to consider both types). These weaknesses are also described in the CWE base, which arranges them into a hierarchy of four levels reflecting abstraction levels. From the CWE base, we automatically gathered the more concrete weaknesses of every previous weakness and their respective mitigations found in a textual section called *CWE mitigations*.

As depicted in Figure 3, we later associate security principles with mitigations by grouping the latter into clusters. It turns out that the section "CWE mitigations" often groups mitigations by categories called Strategies. After a meticulous study of these groups, we observed that they can be associated with security principles without ambiguity. As a consequence, we have directly integrated them as mitigation clusters into the data-store.

The outcome of this systematic extraction is stored in a database DB_2, which encodes relations among 215 attacks, 136 CWE weaknesses, 130 mitigations and 15 clusters. Unsurprisingly, we observed that the attacks having the highest level of abstraction are seldom related to CWE weaknesses, whereas concrete attacks are connected to several mitigation clusters.

The attack CAPEC-39 and its sub-attack CAPEC-31 taken as example, target 18 CWE weaknesses, which illustrate here that the attacks are segmented into more concrete security functionalities. Among them, we have "Improper Input Validation" or "External Control of Critical State Data". These weaknesses can be fixed by 17 mitigations, grouped into 8 clusters.

4.3 Step 3: Security pattern and strong point integration

We manually collected security patterns and their strong points from the catalogue given in [29]. Finding strong points can be a difficult task as these ones are seldom explicitly provided. Strong points often have to be deduced from the sections referring to the forces and intents of the patterns. Afterwards, we manually 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 characterised 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";
2. the second relation is related to the inter-pattern relationships [28]. With P a set of patterns, we define a mapping from P^2 to the annotation set $\{depend, benefit, impair, alternative\}$.

These data and relations, which provide connections among security patterns and strong points, are encoded into the database DB_3. For the domain of Web applications, we gathered 26 security patterns and 36 strong points. For instance, the security pattern "Application firewall" can be characterised with 8 strong points e.g., "Providing the application with a perimeter security mechanism". "Application firewall" is associated with two alternative patterns, "Output Guard" and "Input Guard".

4.4 Step 4: Security principle integration

We chose to organise security principles into a hierarchy, from the most abstract to the most concrete principles. This principle organisation gives a complete hierarchical view on security mechanisms, which are required to cure a weakness and provided by security patterns at the same time. As principles are hierarchically organised, we can link a strong point and a mitigation cluster even if they do not exactly have the same level of abstraction. For instance, consider a strong point and a cluster that are linked to two principles being in different levels of the hierarchy. If one principle is a child of the second one, then the strong point and the cluster will be later related in the classification.

We collected 66 security principles related to Web applications from the papers [20, 25, 11, 5, 10] and manually established dependencies in accordance with the nature of each security principle, often described with text. The resulting hierarchy is certainly not exhaustive but covers the security patterns considered

in the catalogue given in [29]. Figure 5 depicts the security principle hierarchy, which is stored in the database DB_4. There are four levels, the first one being composed of elements labelled by by the most abstract principles, e.g., "Access control".

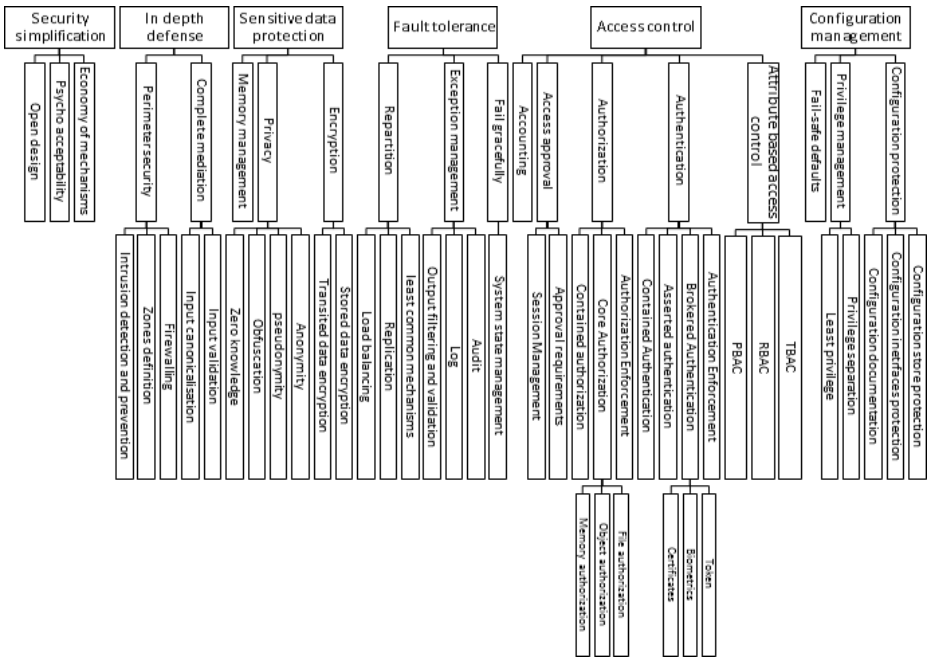


Fig. 5. Security principles organisation

4.5 Step 5: Association among strong points, security principles and mitigation clusters

In this step, we incorporated into the data-store the many-to-many relations between strong points and security principles. We manually performed this step because strong points and principles are mostly presented with different keywords. We observed that the abstraction levels of the strong points better fit with the most concrete security principles, which are labelled in the lowest-level nodes of the hierarchical organisation depicted in Figure 5. But, if a strong point is related to a principle *sp* that is not at the lowest level, then we also link the strong point with all the children of *sp*.

If we take back the example of security pattern "Application Firewall", its strong point "Providing the application with a perimeter security mechanism" can be easily associated with the principle "Perimeter security". As the latter has 3 children in the hierarchy of Figure 5, the strong point is also related to them.

In the same way, we established the many-to-many relations between mitigation clusters and security principles. In Step 3, the clusters include mitigations based upon the same security aspects, e.g., validating user inputs. Once these aspects are deduced, linking clusters and security principles becomes straightforward. For instance, the need for validating user inputs corresponds to the principle "Input validation", which belongs to the principle "Complete mediation" in the security principle hierarchy.

These relations are materialised with the database DB_5, which gathers 15 clusters, 36 strong points and 66 principles.

4.6 Step 6: data consolidation

This automatic step merges the previous databases DB_1 to DB_5 into a single one. On the one hand, DB_1, DB_2, DB_4 and DB_5 store the relations among attacks, weaknesses, mitigations and security principles. On the other hand, DB_3, DB_4 and DB_5 store the relations among security patterns, strong points and security principles. It is now manifest that the security principle hierarchy becomes the central point that helps match attacks with security patterns.

This step is automatically performed with a script by means of the metamodel given in Figure 3. The step produces the final database DB_f, which is available in [17].

5 Security pattern classification and ADTree generation

The database DB_f now holds enough information to organise security patterns and build ADTrees. This section explains how to automatically generate them.

5.1 Security pattern classification

We have chosen to catalogue the combinations of security patterns that could be used to counter an attack stored in DB_f. More precisely, for a given attack Att, we extract:

- the information about the attack (name, identifier, description, etc.),
- the tree $T(Att)$ of attacks, whose root is Att, if Att is not a leaf of the attack tree derived in Step 1.
- for each attack A of $T(Att)$, the hierarchy of security principles $Sp(A)$ by means of the successive relations established among A, weaknesses, clusters and security principles. $Sp(A)$ represents the complete hierarchy of security principles related to an attack, i.e., if a principle sp of $Sp(A)$ is not a leaf of the hierarchical organisation depicted in Figure 5, then we also extract the principle sub-tree whose root is sp;
- for each principle sp in $Sp(A)$, the set of security patterns P_{sp} and the set of patterns $P2_{sp}$ not in P_{sp} that have relations with any pattern of P_{sp}. We also extract the inter-pattern relationships defined for couples of patterns by the relations *depend, benefit, impair, alternative, conflict.*

ID	Security patte	Rela	Relatec	ID	Security pattern	Relationship	Related pattern
39	Application Firewall	-No	-No Value-	31	Application Firewall	alternative	Container Managed Security
		alternative	Input Guard		Authentication Enforcer	benifits	Application Firewall
			Output Guard		Compartmentalization	-No Value-	-No Value-
	Input Guard	-No Value-	-No Value-		Container Managed Security	benifits	Demilitarized Zone
		alternative	Application Firewall		Controlled Object Factory	-No Value-	-No Value-
		benifits	Output Guard		Input Guard	benifits	Authentication Enforcer
	Pathname Canonicalization	-No Value-	-No Value-		Output Guard	benifits	Secure Service Facade
		alternative	Compartment		Pathname Canonicalization	-No Value-	-No Value-
					Secure Pipe	-No Value-	-No Value-
					Secure Service Facade	benifits	Demilitarized Zone

Fig. 6. Data extraction for the attack CAPEC-39

Figure 6 depicts an extraction example for the attack CAPEC-39. The first column gives the attack identifier. The next column gives the security pattern allowing to counter the attack. Columns 3 and 4 provide the inter-pattern relations, e.g., "Application Firewall" is an alternative to "Input Guard". The attack CAPEC-39 has one sub-attack CAPEC-31, whose identifier is provided in Column 5. The three last columns give the security patterns allowing to overcome the attack CAPEC-31 and their relations with other patterns.

The data extraction is automatically performed with a tool based upon Talend. Once the tool has covered all the attacks stored in the database DB_f, we obtain the security pattern classification. This tool can be re-executed every time the data-store is updated. The classification remains up-to-date accordingly.

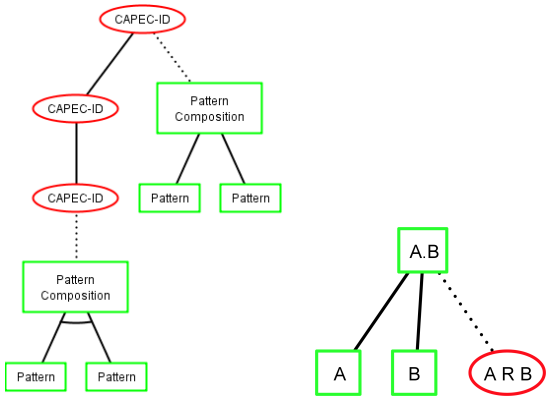
Unfortunately, we think that Comprehensibility, which refers to the ability to use the classification by experts or novices, is not yet totally satisfied at this stage. Indeed, the classification is given under a tabular form, which does not appear to be the most user-friendly way to represent a classification. This is why we also propose to improve its readability with ADTrees.

5.2 Attack-Defense Tree generation

ADTrees *are graphical representations of possible measures an attacker might take in order to attack a system and the defenses that a defender can employ to protect the system* [9]. ADTrees have two different kinds of nodes: attack nodes (red circles) and defense nodes (green squares). A node can be *refined* with child nodes and can have one child of the opposite type (linked with a dashed line). Node refinements are either disjunctive or conjunctive. The former is recognisable by edges going from a node to its children. The latter is graphically distinguishable by connecting these edges with an arc.

We generate ADTrees having the general form illustrated in Figure 7(a). The root of this tree is labelled by an attack. If the latter has sub-attacks, these are given in the tree with children linked with a disjunctive refinement and so forth. Furthermore, the tree points out how to prevent attacks with defenses given

under the form of security pattern combinations. A defense node is linked to an attack node with a dashed line. This defense node is either labelled by a security pattern, or is the root of a sub-tree showing patterns and their relations. We generate ADTrees with the following steps:



(a) Pattern classification representation with ADTrees

(b) Conflicting pattern representation with ADTree

Fig. 7. ADTree examples, reprinted from A methodology of security pattern classification and of attack-defense tree generation, by Regainia, L., Salva, S., Proceedings of ICISSP'17, SciTePress, Porto, Portugal (02 2017).

1. every attack found in DB_f has its own ADTree whose root node is labelled by its identifier. This root node is linked to other attack nodes with a disjunctive refinement if the attack has sub-attacks. This step is repeated for every sub-attack. In other words, we generate a sub-tree of the original hierarchical tree extracted in Step 1, whose root is an attack;
2. for every attack node A, we collect the set P of security patterns that counter the attack. The inter-pattern relationships are illustrated in the ADTree with new nodes and logic operations. Given a couple of patterns $(p_1, p_2) \in P$, if we have:
   - $(p_1\ R\ p_2)$ with R a relation in $\{depend, benefit\}$, we build three defense nodes, one parent node labelled by R and two nodes labelled by p_1, p_2 combined with this parent defense node by a conjunctive refinement;
   - $(p_1\ alternative\ p_2)$, we build three defense nodes, one parent node labelled by $alternative$ and two nodes labelled by p_1, p_2, which are linked by a disjunctive refinement to the parent node;
   - $(p_1\ R\ p_2)$ with R a relation in $\{impair, conflict\}$. In this particular case, we would want to use the xor operation since both patterns can be used but the implementation of p_2 decreases the efficiency or conflicts with p_1. Unfortunately, this operation is not available with this

tree model. Therefore, we replace the operator by the classical formula $(A\ xor\ B) \longrightarrow ((A\ or\ B)\ and\ not\ (A\ and\ B))$. The *not* operation is here replaced by an attack node meaning that two conflicting security patterns used together constitute a kind of attack. The corresponding subtree is depicted in Figure 7(b),

– p_1 having no relation with any pattern p_2 in P, we add one parent defense node labelled with p_1.

The parent defense nodes, resulting from the above steps, are combined to a defense node labelled by "Pattern Composition" with a conjunctive refinement. This last defense node is linked to the attack node A.

When an attack is linked to several security patterns, the second step can achieve a large defense sub-tree. But, this one can often be simplified. In short, if we replace the relations *depend*, *benefit* by the operation AND, the relation *alternative* by OR and the relations *impair*, *conflict* by XOR, we obtain logical expressions. These expressions can be reduced with tools, e.g., BExpRed[4]. A simplified defense tree can be derived from the reduced expression. For instance, with the three patterns p1, p2 and p3 having the relations (p1 *benefit* p2), (p1 *alternative* p3) and (p2 *alternative* p3), we obtain (p1 AND p2) AND (p2 OR p3) AND (p1 OR p3), which can be reduced by (p1 AND p2). This expression gives a defense node that is conjunctively refined with two nodes labelled by p1 and p2.

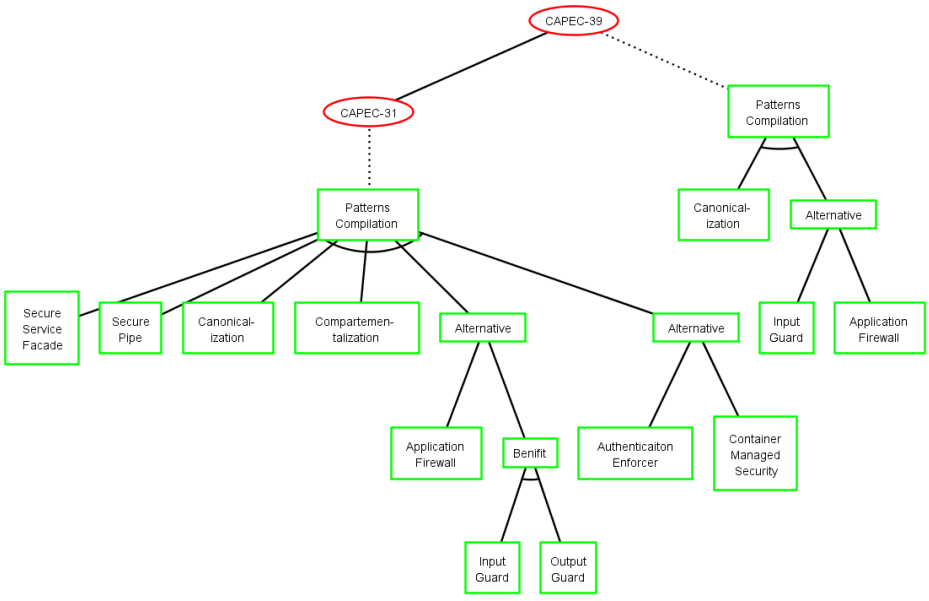


Fig. 8. ADTree of the attack CAPEC-39

[4] `https://sourceforge.net/projects/bexpred/`

We implemented the ADTree generation with a tool, which takes as input an attack identifier and yields an ADTree, which is stored into an XML file. These files can be used with the ADTree editing tool ADTool presented in [8]. As a consequence, ADTrees can be modified or updated as the designer wishes.

If we take back the attack CAPEC-39, we obtain the ADTree of Figure 8. This tree firstly shows that the attack CAPEC-39 has the sub-attack CAPEC-31. Both attacks can be countered by several security pattern combinations. For instance, the attack CAPEC-39 can be countered by two pattern combinations: the pattern "Canonicalization" must be used either with "Application Firewall" or with "Input guard" since both are alternative patterns. The number of security patterns related to the attacks CAPEC-39 and CAPEC-31 is explained here by the diversity of the targeted weaknesses. Indeed, 18 weaknesses can be exploited here (6 for the attack CAPEC-39 and 12 for CAPEC-31). We assume for the classification generation that all of them have to be mitigated. As they cover different security issues, e.g., input validation problems, privilege management or encryption problems, several patterns are required to fix the weaknesses and hence block the attacks.

This example illustrates that a designer can follow the concrete materialisations of an attack in an ADTree. He/she can choose the most appropriate attack with respect to the context of the application being designed. The ADTree provides the different security pattern combinations that have to be used to prevent this attack.

6 Empirical Evaluation

In order to assess whether designers can take profit of our classification and ADTrees, we empirically studied two scenarios where 25 participants were given the task of choosing security pattern combinations to prevent two attacks, CAPEC-244: Cross-Site Scripting via Encoded URI Schemes and CAPEC-66: SQL Injection, on two vulnerable Web applications, *Ropeytasks* [5] and *Bodgeit* [6]. The participants are third to fourth year computer science undergraduate students; most of them have good skills in the design, development and test of Web applications. They have some knowledge about classical attacks and are used to handle design patterns, but not security patterns. The duration of each scenario was set at most to one hour.

In the first scenario, denoted Part 1, we supplied these documents to the students: the CAPEC base, two concrete examples showing how to perform each attack, the catalogue of security patterns given in [29] and the pattern classification proposed in [1]. For simplicity, we refer to these documents as basic pattern documents in the remainder of the evaluation. In the second scenario, denoted Part 2, we supplied additional documents for the two attacks, i.e., our classification under the form of tabulars, two ADTrees generated from the data-

[5] https://github.com/continuumsecurity/RopeyTasks
[6] https://github.com/psiinon/bodgeit

store. At the end of each scenario, the students were invited to fill in a form listing these questions:

- Q1: Was it difficult to choose security patterns?
- Q2: Was it difficult to use the CAPEC documentation (in Part 1) / our classification+ADTrees (in Part 2)?
- Q3: Was it difficult to use the basic pattern documents (in Part 1) / our classification+ADTrees (in Part 2)?
- Q4: What was your time spent for choosing security patterns?
- Q5: How confident are you in your pattern choice?
- Q6: What are the patterns you have chosen?

This form was actually devised to evaluate the following criteria:

- C1: Comprehensibility: does our classification make the pattern choice less difficult?
- C2: Efficiency: does our classification help reduce the time needed to choose patterns?
- C3: Accuracy: are the chosen patterns correct ?

6.1 Experiment results

From the forms returned by the participants (available in [17]), we extracted the following results. Firstly, Figure 9 illustrates the percentages of answers to the questions Q1 to Q3. For these, we used this four-valued scale: *easy, fairly easy, difficult, very difficult*. From Question Q4, we collected the time spent by the participants for choosing patterns (in Part 1 and 2 of the experimentation). In summary, response times varied between 15 and 50 minutes for Part 1, and between 5 and 30 minutes for Part 2. The bar charts of Figure 10 depicts the levels of confidence of the participants towards their security pattern choices (Question Q5). The possible answers were for both scenarios: *very sure, sure, fairly sure, not sure*.

We finally analysed the security pattern combinations provided by the participants in Question Q6. We organised these responses into four categories (ordered from the more to the less accurate):

- Correct: several pattern combinations were accurate. When a participant gives one of these combinations, its response belongs to this category;
- Correct+Additional: this category includes the responses composed of a correct pattern combination, completed with some other patterns;
- Missing: we gather in this category, the incomplete pattern combinations without additional patterns;
- Missing+Additional: this category holds the worst responses, composed of unexpected patterns eventually accompanied with some expected ones.

With these categories, we obtained the bar charts of Figure 11, which gives the number of responses per category and per experiment scenario.

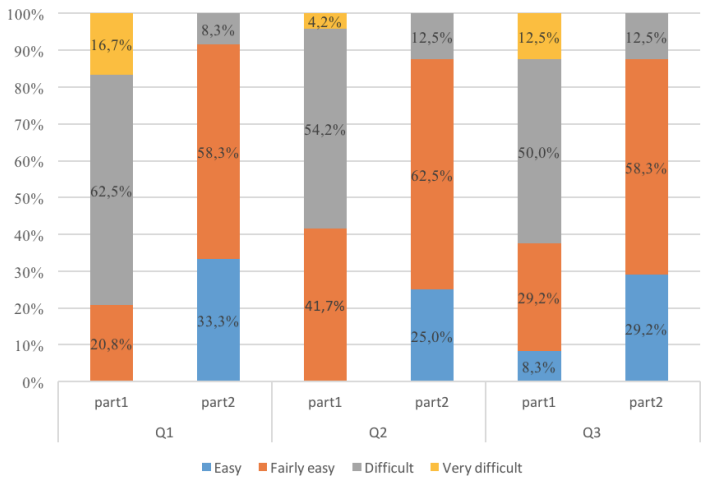


Fig. 9. Response rates for Q1 to Q3

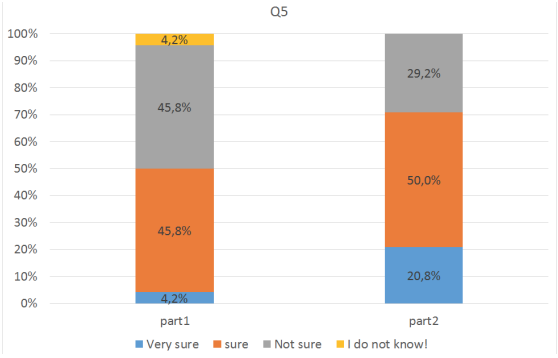


Fig. 10. Confidence rates (Q5)

6.2 Result interpretation

C1 Comprehensibility: Figure 9 shows that 33% of the participants estimated that the pattern choice was easy with our classification and ADTrees (Q1). In contrast, no participant found that the choice was easy when using only the basic pattern documents. The rate of "Easy" Fairly Easy" increased by 70,8% between Part 1 and Part 2. With Question Q2, 41,7% of the participants found "Fairly easy" the use of the CAPEC base, whereas 87,5% esteemed our documents (ADTrees) "Easy" and "Fairly Easy" to use. Similarly, only 37,5% of the participants found "Easy" and "Fairly easy" the reading of the basic pattern documents. This rate reaches 87,5% with our classification. Consequently, Figure 9 shows that our classification and ADTrees make the pattern choice easier and that they are simpler to interpret than the basic pattern documents. Figure 10

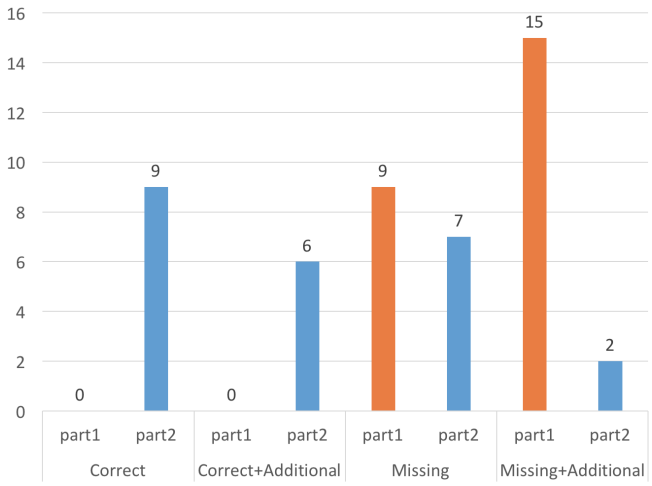


Fig. 11. Accuracy Measurement (Q6)

expresses that the confidence of the participants on their responses increased by 20,8 %.

C2 Efficiency: the average time spent by the participants for choosing patterns is equal to 32 minutes in the first scenario (Part 1). This time delay decreases to 15 minutes when the participants employed our classification and ADTrees. Furthermore, no participants went over 30 minutes for choosing patterns in Part 2 (in contrast with 50 minutes for Part 1). Hence, our documents make the participants more efficient.

C3 Accuracy: Figure 11 reveals how complicated it is to read the basic pattern documents. Indeed, no participant gave a correct pattern combination in Part 1. In contrast, when they used our classification and ADTrees, the number of correct responses rises to 15 out of 25 (60%). Furthermore, the category of responses "Missing+Additional" (worst responses) is strongly reduced (60 % with Part 1 to 8% with Part 2). Consequently, the pattern choice is significantly more accurate with our classification and ADTrees. Nonetheless, even with our documents, the number of participants that gave incomplete pattern combinations remains around the same range (9 in Part 1, 7 in Part 2). More efforts seem required to avoid the participants forgetting patterns in ADTrees.

7 Classification Discussion

Our current classification is built on a non exhaustive set of 215 CAPEC attacks, 26 security patterns and 136 CWE weaknesses related to Web applications. Presented in a tabular form, it enables multi-attribute based decisions insofar as patterns can be classified according to security principles, weaknesses and attacks. The classification complies with seven of the nine quality criteria defined in [2] :

- Navigability: our classification, accompanied by ADTrees, satisfies this criterion as it exposes the hierarchical refinements of an attack and the combinations of patterns, which should be integrated in the application model. In addition, the classification provides the relationships among security patterns, which help choose the most appropriate pattern combination. For instance, if two conflicting patterns are listed, the classification points out this conflict to avoid using them together;
- Determinism: the classification is clearly defined by means of the integration steps. These justify the soundness of the classification;
- Unambiguity/Comprehensibility: as patterns are classified w.r.t. attacks and security principles, we provide a clear category structure. This organisation, which is supplemented and illustrated by means of ADTrees, makes our classification readable and comprehensible even for novices in security patterns;
- Usefulness: we believe the classification can be used in practice since it is based upon the security pattern catalogue given in [29] and the CAPEC and CWE bases. Furthermore, the Attack tree formalism is one of the most prominent security formalism for analysing threats. The ADTree model is supported by several tools, in particular ADTool [8]. Our ADTree generator actually generates XML files taken as inputs by ADTools;
- Acceptability: *an acceptable classification schema should be structured in a way that it provides help in partitioning the security pattern landscape and becomes generally approved* [2]. Our classification partitions security patterns with regard to attacks, weaknesses and security principles. Furthermore, our evaluation shows that the classification makes participants more efficient and confident on their pattern choices without providing new constraints;
- Repeatability: the classification is generic and can be reused. Furthermore, the data-store and the classification can be updated.

In our classification, a security pattern can be related to several attacks and security principles. As a consequence, it is not Mutual exclusive (patterns should be categorised into at most one category). Even though it is not a primary goal of our classification, we could fix this issue by grouping attacks into contexts in a mutual way, like in [4]. To do so, the meta-model of Figure 3 should be updated with a new entity called Context linked to the entity Attack. Like most of the pattern classifications, the Completeness criterion is not met as we do not yet consider all the available security patterns.

We compared our classification with the two papers providing relations between security patterns and attacks [27, 1]. In these works, the security pattern intents are manually compared to the summaries of the attacks. As these textual sections are abstract, few relations were found. The largest contribution is provided by Alvi et al. who considered around 20 attacks and manually linked them to 5 patterns. In contrast to these works, our classification is more complete: we provide 26 security patterns as solutions against 215 attacks of the CAPEC base. Our classification exposes more pattern combinations per attack; the more choice is not always the better though. After inspection, we observed that more than one or two patterns are generally required to counter attacks. A

last important point is that the classifications exposed in [27, 1] do not contradict our relations between attack and patterns. For instance, the attack "CAPEC-66 SQL Injection" is related to the security patterns "Intercepting Validator" and "Input validation" in [27]. The attacks "CAPEC-244: Cross-Site Scripting via Encoded URI Schemes" and CAPEC-66 are only associated with the pattern "Intercepting Validator" in [1]. For these attacks, our method generates two ADTrees, which provide 4 combinations of 7 patterns for the CAPEC-244 and 8 combinations of 9 patterns for the CAPEC-66. As in [27, 1], the ADTrees exhibit the pattern "Input Guard", which can be implemented by "Intercepting Validator". But, they also list other patterns. For the CAPEC-244, some of these patterns are alternative to "Input Guard", e.g., "Application Firewall". Other patterns, e.g., "Authentication Enforcer" or "Controlled Object Monitor" are related to specific weaknesses targeted by the attack CAPEC-244. We believe these patterns, which are not given in the previous classifications, are required to counter the attack with regard to the application context.

Some statistical information can be automatically extracted from our classification, e.g., the ratio of weaknesses to attacks, of patterns to attacks. For instance, Figure 12 shows the number of attacks at least partially countered per pattern. Keeping in mind, that the current set of patterns is not exhaustive, we observe that 2 patterns seem to emerge for partly fixing a large part of the 215 attacks covered by the classification: "Input Guard" and "Application firewall", can overcome 113 and 109 attacks respectively. This kind of information can guide designers towards security analysis and good practices. For instance, with the above chart and ADTrees, a designer can deduce that the patterns "Input Guard" and "Application firewall" are alternative security patterns and that one of them should be used in the design of Web applications as they partially block numerous attacks. It is manifest that if we complete the data-store with more data, e.g., attack risks, such charts could be more refined and adapted to the developer needs.

Limitations: Our classification and method present some limitations, which could lead to some research future work:

- we did not envisaged the notion of attack combination. Such a combination could be seen as several attacks or as one particular attack. If an attack combination can be identified and documented with its sub-attacks, then it can be integrated in our data-store;
- the ADTree size limit is not supported by our ADTree generator. When an attack has a high level of abstraction, we observed that the resulting ADTree size can become large because it includes a set of sub-attacks, themselves linked to several patterns. This is a strong limitation since large trees are usually unreadable, which contradicts the method purposes;
- the classification is not exhaustive: it includes 215 attacks out of 569 (for any kind of application), 210 CWE weaknesses out of around 1000 and 26 security patterns out of around 176. It can be completed with new attacks automatically. But it worth mentioning that the completion of the data-

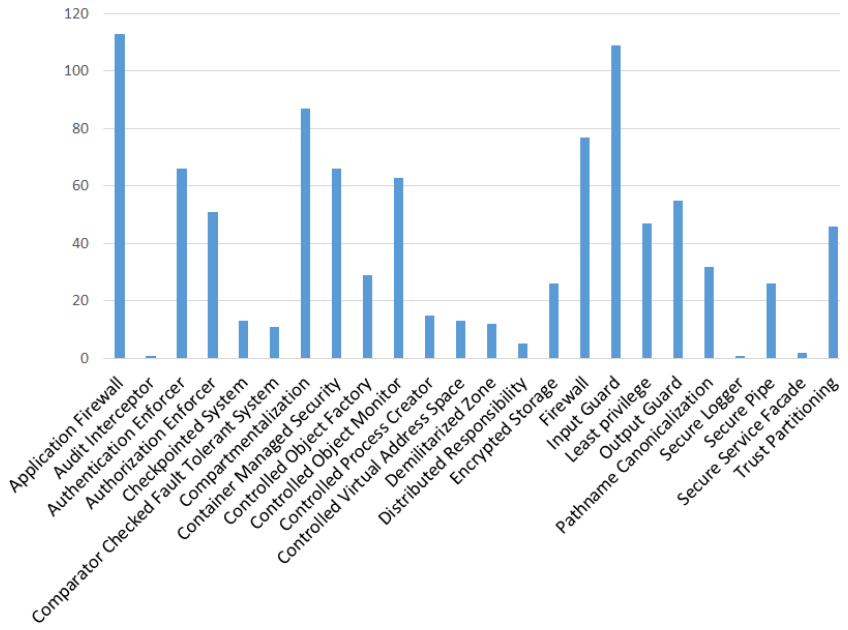


Fig. 12. Number of fixed attacks per pattern, reprinted from A methodology of security pattern classification and of attack-defense tree generation, by Regainia, L., Salva, S., Proceedings of ICISSP'17, SciTePress, Porto, Portugal (02 2017).

store with new security patterns or weaknesses requires some manual steps. It could relevant to investigate whether some text mining techniques would help partially automate these manual steps. The classification exhaustiveness also depends on the available security data. In the ADTree of Figure 8, all the attack nodes are linked to defense nodes. Sometimes, with other attacks, no defenses are provided. This can be generally explained by three main reasons: 1. the attack is too abstract to be associated with weaknesses. This attack should be linked to sub-attacks though; 2. Security databases or pattern catalogues are incomplete (lack of mitigation, weakness, etc.). More data are required while the data integration process; 3. the attack is relatively new and cannot be yet overcame by security patterns;

– several steps require manual interventions, which are prone to errors. These manual steps may lead to associations among security data that are bound to be controversial. We compared our results with other papers, but this is insufficient to ensure all the associations are correct or that no security data e.g., strong point, is missing. Validating every relation is a hard problem. It could be partially solved by the use of verification methods. But the writing of formal expressions for modelling the entities and associations of our meta-model is also a long and error-prone task.

8 Conclusion

The generic nature of security patterns and their growing number make their choice difficult for overcoming a security problem. This is why we have presented a security pattern classification method putting together CAPEC attacks, CWE weaknesses and security patterns to guide designers in their pattern choices. This method provides a meta-model and the data integration steps to generate a pattern classification showing the patterns that can be used to counter an attack. Pattern intern-relationships are also given to increase Navigability and Comprehensibility. The method automatically generates ADTrees, which ease the classification readability. These ADTrees could be taken as a first step of other security processes, e.g., threat modelling.

In future research, we firstly intend to focus on the automation of some of the data integration steps. We will investigate whether some text mining techniques would help partially automate the extraction and integration of security data without bringing ambiguity. Our method does not take into consideration the size of the ADTrees. The ADTree reduction could be a first solution on this problem. But, the literature does not yet provide a generic method for this kind of reduction. Reducing such trees remains a hard problem as the node meaning must be taken into account in the node aggregating process. We intend to investigate on this issue in future works.

References

1. Alvi, A.K., Zulkernine, M.: A Natural Classification Scheme for Software Security Patterns. 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing pp. 113–120 (2011)
2. Alvi, Aleem, K., Zulkernine, M.: A Comparative Study of Software Security Pattern Classifications. 2012 Seventh International Conference on Availability, Reliability and Security pp. 582–589 (2012)
3. Anand, P., Ryoo, J., Kazman, R.: Vulnerability-Based Security Pattern Categorization in Search of Missing Patterns. 2014 Ninth International Conference on Availability, Reliability and Security pp. 476–483 (2014)
4. Bunke, M., Koschke, R., Sohr, K.: Organizing security patterns related to security and pattern recognition requirements. International Journal on Advances in Security 5 (2012)
5. Dialani, V., Miles, S., Moreau, L., De Roure, D., Luck, M.: Transparent fault tolerance for web services based architectures. In: Euro-Par 2002 Parallel Processing, pp. 889–898. Springer (2002)
6. Fernandez, E.B., Washizaki, H., Yoshioka, N., Kubo, A., Fukazawa, Y.: Classifying security patterns. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 4976 LNCS, pp. 342–347 (2008)
7. Harb, D., Bouhours, C., Leblanc, H.: Using an Ontology to Suggest Software Design Patterns Integration, pp. 318–331. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-01648-6_34
8. Kordy, B., Kordy, P., Mauw, S., Schweitzer, P.: ADTool: Security Analysis with Attack–Defense Trees, pp. 173–176. Springer, Berlin, Heidelberg (2013)

9. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack–defense trees. Journal of Logic and Computation p. exs029 (2012)
10. Meier, J.: Web application security engineering. Security & Privacy, IEEE 4(4), 16–24 (2006)
11. Meier, J., Mackman, A., Dunner, M., Vasireddy, S., Escamilla, R., Murukan, A.: Improving web application security: threats and countermeasures. Microsoft Corporation 3 (2003)
12. Mitre corporation: Common attack pattern enumeration and classification (2017), https://capec.mitre.org/
13. Mitre corporation: Common weakness enumeration (2017), https://cwe.mitre.org/
14. Munawar, H.: Security pattern catalog, http://www.munawarhafiz.com/securitypatterncatalog/
15. OWASP: The open web application security project (owasp). In: http://www.owasp.org (2017)
16. Regainia, L., Salva, S.: A methodology of security pattern classification and of attack-defense tree generation. In: Camp, O., Furnell, S., Mori, P. (eds.) Proceedings of the 3nd International Conference on Information Systems Security and Privacy (ICISSP 2017. SciTePress, Porto, Portugal (02 2017)
17. Regainia, L., Salva, S.: Security pattern classification, companion site, http://regainia.com/research/companion.html
18. Regainia, L., Salva, S., Bouhours, C.: A classification methodology for security patterns to help fix software weaknesses. In: Proceedings of the 13th ACS/IEEE International Conference on Computer Systems and Applications AICCSA (2016)
19. Rodriguez, E.: Security Design Patterns. In: 19th Annual Computer Security Application Conference (ACSAC'03) (2003)
20. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE 63(9), 1278–1308 (1975)
21. Schumacher, M., Roedig, U.: Security Engineering with Patterns. Engineering 2754, 1–208 (2001), http://www.springerlink.com/content/y01ll29crpx0l8yu
22. Slavin, R., Niu, J.: Security patterns repository (2016), http://sefm.cs.utsa.edu/repository/
23. Tøndel, I.A., Jensen, J., Røstad, L.: Combining misuse cases with attack trees and security activity models. In: Availability, Reliability, and Security, 2010. ARES'10 International Conference on. pp. 438–445. IEEE (2010)
24. Uzunov, A.V., Fernandez, E.B.: An extensible pattern-based library and taxonomy of security threats for distributed systems. Computer Standards & Interfaces 36(4), 734–747 (2014)
25. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way, Portable Documents. Pearson Education (2001)
26. Wassermann, R., Cheng, B.H.: Security patterns. In: Michigan State University, PLoP Conf. Citeseer (2003)
27. Wiesauer, A., Sametinger, J.: A security design pattern taxonomy based on attack patterns. In: International Joint Conference on e-Business and Telecommunications. pp. 387–394 (2009)
28. Yskout, K., Heyman, T., Scandariato, R., Joosen, W.: A system of security patterns (2006)
29. Yskout, K., Scandariato, R., Joosen, W.: Do security patterns really help designers? In: Proceedings of the 37th International Conference on Software Engineering - Volume 1. pp. 292–302. ICSE '15, IEEE Press, Piscataway, NJ, USA (2015)