A catalogue associating security patterns and attack steps to design secure applications

Sébastien Salva, Loukmen Regainia

A catalogue associating security patterns and attack steps to design secure applications.

Sébastien SALVA [a,1], Loukmen REGAINIA [b]

[a] *University Clermont Auvergne, IUT of Clermont-Ferrand, LIMOS, FRANCE, E-mail: sebastien.salva@uca.fr.*

[b] *University Clermont Auvergne, LIMOS, FRANCE, E-mail: loukmen.regainia@uca.fr.*

Abstract. Design Patterns are now widely accepted and used in software engineering; they represent generic and reusable solutions to common problems in software design. Security patterns are specialised patterns whose purpose is to help design applications that should meet security requirements. The enthusiasm surrounding security patterns has made emerge several catalogues listing up to 180 different patterns at the moment. This growing number brings an increased difficulty in choosing the most appropriate patterns for a given design problem. We propose a security pattern classification to facilitate the security pattern choice and a classification method based on data integration. The classification exposes relationships among software attacks, security principles and security patterns. It expresses the pattern combinations that are countermeasures to a given attack. This classification is semi-automatically inferred by means of a data-store integrating disparate publicly available security data. The data-store is also used to generate *Attack Defense Trees*. In our context, these illustrate, for a given attack, its sub-attacks, steps, techniques 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 human subjects the benefits of using a pattern classification established for Web applications, which covers 215 attacks, 66 security principles and 26 security patterns.

Keywords. Attack, Security patterns, Classification, Security principles, Attack Defense Tree

Introduction

The World Wide Web is continuously expanding with new unstructured or semi-structured information, especially since the recent explosion of digitalisation of almost everything e.g., music, books, encyclopaedia, documents, etc. In the domain of software security, many documents, knowledge bases or papers are now publicly available to help engineers develop secure applications. These numerous digitalised resources are presented with different viewpoints (attackers, defenders, etc.), formats (text, database,

[1] Corresponding Author: University Clermont Auvergne, IUT of Clermonf-Ferrand LIMOS, F-63000 CLERMONT-FERRAND, FRANCE; E-mail: sebastien.salva@uca.fr.

etc.), abstraction levels (security principles, attack steps, exploits, etc.) or contexts (system, network, etc.). Furthermore, these different documents are meaningful at different stages of the software life cycle. For instance, the Exploit base [18] gathers scripts that can be used to test whether an application is vulnerable. In another context, the notion of security patterns, which is one of the topics of this paper, aims at providing guidelines to help design secure systems [36,27,24,8]. 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* [24]. Schumacher also postulates that *Security patterns relates countermeasures to threats and attacks in a given context* [27].

This plethora of (often complex) documents makes increasingly harder for a designer to select the appropriate solution in a given context. Indeed, they cannot be experts in all fields and they usually lack guidance for conceiving secure software or systems.

Several authors recently focused on security patterns to better guide designers. Security patterns were organised along the following categories: by security principles [37,2], by application domains [5] (software, network, user, etc.), by vulnerabilities [3,1] or by attacks [34,1]. Despite the benefits brought by these classifications, they suffer from several limitations, which prevent their adoptions in the industry. Firstly, these classifications were manually devised, by comparing directly textual descriptions of different security concepts (patterns, principles, vulnerabilities, attacks, etc.). As these descriptions are generic and have diverse abstraction levels, the categorisation of a pattern can only be performed when there is an evident relationship with a security property. In addition, as these classifications are not *deterministic* (no strict definition of the classification process [2]), it becomes often delicate to upgrade them. Yskout et al. also reported that the security pattern adoption is limited *possibly due to a sub-optimal quality of the documentation* [38]. We indeed believe that security pattern classifications lack Navigability and Comprehensibility, which are quality criteria, proposed in [2] and respectively related to: the ability to direct a software designer among collaborative and related patterns; the ease to understand patterns by both a novice and an expert developer.

From these observations, we propose in this paper a method for classifying security patterns based on the concept of data integration, namely we combine security data coming from different sources and provide an unified view on this data. To make this classification navigable and comprehensible, we propose to automatically infer *Attack Defense Trees* [13], which illustrate the security pattern combinations that can be used to prevent an attack. More precisely, our contributions are as follows.

- We present a data-store meta-model and a data integration method consisting of six steps, which extract data from various publicly accessible sources and store relationships among attacks, security principles and security patterns. The method breaks down security properties into sub-properties and associates them to achieve a precise classification.
- Our security pattern classification is automatically derived from this data-store. For an attack, the classification expresses the pattern combinations that can be integrated in the application model to later prevent the attack from being successfully carried out.
- We automatically generate Attack-Defence Trees (ADTrees) which aim at supplementing the classification with illustrations depicting, for a given attack, its (more concrete) sub-attacks, steps and techniques along with defenses expressed with

security patterns. These ADTrees aim at improving the understanding of the previous classification. They can also be used as security requirement documents for threat risk modelling.

As a proof of concept, we have generated a data-store and a security pattern classification specialised to the Web application domain. The classification is composed of 215 CAPEC attacks, 26 security patterns and 66 security principles covering various security aspects. We also provide a tool to generate ADTrees. This classification and the ADTree generator are available in [21]. We employed them to evaluate, on 24 human subjects, the benefits of using our pattern classification and ADTrees with regard to the following criteria: Comprehensibility, Effectiveness and Accuracy.

The remainder of the paper is organised as follows. We present in Section 1 the related work and the motivations of our approach. Besides, we introduce some security notions and data used throughout the paper. The method used to integrate security data and to build a data-store, is given in Section 2. Section 3 shows how the pattern classification and ADTrees are automatically extracted from the data-store. We list the quality criteria met by the classification and discuss its limitations in Section 4. We evaluate the classification and ADTrees in Section 5. We also discuss about the threats to the validity of the evaluation. We finally conclude and give some perspectives for future work in Section 6.

1. Background

1.1. Related Work

Several security pattern catalogues are available in the literature [28,17,38], gathering a total of 176 patterns. These growing catalogues make difficult the choice of the appropriate patterns for overcoming a security problem.

Many classifications were proposed in the literature to ease the pattern choice with regard to a given context. The classifications proposed in [34,30,1,31] 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 software designers. Initially, Wiesauer et al. presented in [34] a short taxonomy of security design patterns made from links between attack textual descriptions and security pattern purposes. Tondel et al. presented in [30] 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 of 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 natural 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 an augmented template composed of the essential elements needed for designers. They manually completed 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. Uzunov et al. introduced in

[31] 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 their “threat patterns” are abstract enough to encompass security problems related to the context of distributed systems.

In the papers [2,5], the authors discuss some limitations exposed by pattern classifications. Alvi et al. outlined 24 pattern classifications, including security pattern classifications, and established a comparative study to point out their positive and negative aspects [2]. They chose 29 classification attributes (purpose, abstraction levels, life-cycle, etc.) and compared the classifications against a set of desirable quality criteria (Navigability, Comprehensibility, Usefulness, etc.). They observed that several classifications were built w.r.t. 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. 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 compared design pattern and security pattern classifications [5]. They finally proposed a classification based on the application domains of patterns (software, network, user, etc.). Yskout et al. also reported that the security pattern adoption is limited *possibly due to a sub-optimal quality of the documentation* [38].

After reviewing these classifications, we indeed believe that security pattern classifications lack Navigability and Comprehensibility. We also observed these are all manually conceived by interpreting different documents to find abstract relationships. Justifying these classifications or extending them is often difficult. Furthermore, the relations among patterns are often not given, yet we noticed that some patterns are compatible together and that others are conflicting. As a consequence, a designer may be still confused about the pattern choice. As in [1], we propose a pattern classification expressing which security patterns can be used to prevent an attack step from being successfully executed on an application (and hence an attack, even though a step is more precise). Our classification proposes a more precise and accurate mapping between patterns and attacks. It is more accurate in the sense that we translate the meaning of the patterns and attacks into smaller properties, i.e., strong points, attack steps, techniques, and countermeasures. We establish relations among these properties with respect to security principles, which identify the meaning of these relations. In addition, the classification is completed with the inter-pattern relationships found in [37]. This is why we claim that our classification is more precise. Another contribution of this paper lies in the presentation of a classification process based on data integration. This one includes six manual and automatic steps, which offer the advantage of justifying the soundness of the pattern classification and reduce the efforts required to add new patterns or attacks to the classification. Finally, we complete the classification with ADTrees making the classification readable even for novice in patterns or security.

We now compare this paper with some of our own previous work. Initially, we proposed in [22] a security pattern classification grouping patterns according to the weaknesses that they can cure. That paper can be seen as a first step toward the approach presented here. We indeed exploit the weaknesses listed in the CWE base [16] to categorise patterns. However, unlike the present paper, we did not focused on attacks or attack trees. In [20], we have completed the previous method this time to organise security patterns in relation to attacks. Attacks are collected from the CAPEC base as the first step of the method developed in this paper. Then, attacks are associated to weaknesses, themselves linked to patterns. The difference with the present paper firstly lies in the data integration process. We do not consider weaknesses but other kinds of security properties. We indeed consider that an attack is composed of techniques and sequential steps, which we associate to countermeasures. We semi-automatically groups them with a text mining approach. The resulting clusters are finally associated to patterns. These relations are strictly modelled by a new meta-model presented in this paper, which structures our data-store. We also generate ADTrees illustrating some relations of the meta-model, in particular the sequences of attack steps. Furthermore, we present an experimentation performed on 24 participants to estimate the benefits of using our classification.

1.2. Publicly Accessible Resources For The Data Integration

We present below the publicly accessible resources (documents, databases, research papers) we studied to devise a data-store for the security pattern classification generation.

1.2.1. Security pattern documents

We firstly recall that security patterns provide guidelines for secure system design and evaluation [36,8]. Generally, they are presented textually or with schema, e.g., UML diagrams, and are characterised by a set of structural and behavioural properties. Schumacher defines more precisely security patterns as triples $P := (C, P, S)$ where C is a security context, P a security problem and S a security solution [27].

Several security pattern catalogues are available in the literature [28,17,38], themselves extracted from other papers. In these documents, a security pattern is usually characterised with its solutions (a.k.a. intents), its interests called forces and the consequences of applying the pattern to an application. The quality of a pattern and its classification can be established by means of its *strong points*, which are sub-properties of the pattern [10] related to its features. Strong points are manually extracted from the forces and consequences of a security pattern, given in its description.

In addition, a security pattern can be documented to express its relationships with other patterns. These properties may noticeably help combine patterns and not to devise unsound composite patterns. Yskout et al. proposed the following annotations between two patterns p_1 and p_2 [37]:

- ”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.

“Secure Logger” is a security pattern example whose primary objective is to store application events in a centralised way so that it should be impossible to alter log files. Figure 1 depicts a class diagram of this security pattern. This schema implies that “Secure

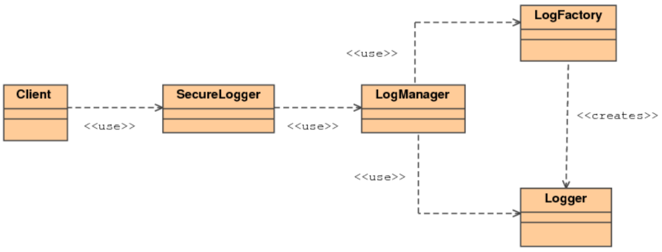


Figure 1. Class layout of the security pattern "Secure Logger"

Logger" provides a mean of decoupling the implementation details of the logger from the remainder of the application. This corresponds to a strong point of the pattern. Its strong points are summarised below:

1. logs sensitive information that should not be accessible to unauthorized users;
2. ensures the integrity of the logged data to determine if it was tampered with by an intruder;
3. captures output at one level for normal operations and at other levels for greater debugging in the event of a failure or an attack;
4. centralises control of logging in the system for management purposes;
5. code must be adaptable and extensible to protect against both current and future threats;
6. performs all of the necessary security processing prior to the actual logging of the data, which allows management of each function independently of the others without the risk of impacting overall security.

This pattern can be implemented by means of two other security patterns [38]: "Audit Interceptor" or "Secure Pipe". The former may be used to collect the events that are stored by "Secure Logger". The latter may be used to guarantee that the data is not tampered with in transit to a secure store.

1.2.2. The CAPEC base

The Common Attack Pattern Enumeration and Classification (CAPEC) is an open database offering a catalogue of attacks in a comprehensive schema [15]. Attack patterns are descriptions of common approaches that attackers might take to attack software or systems. An attack pattern, which we refer here as documentation (to avoid the confusion with security pattern), consists of several sections. The section "Related attack patterns" shows interdependence among attacks having different levels of abstractions. The first two levels (denoted Category and Meta pattern) give attack mechanisms, the last two levels (Standard pattern and Detailed attack pattern) details attacks.

Different binary relations are given between two attacks. Among them, we noted:

- a is member of/child of b: when the attack a is a refinement of the attack b,
- a has member/parent of b: when the attack a is more abstract than b.

Besides, the attacks of the last two levels have specific paragraphs to describe other properties, e.g., impact, prerequisites, severity, required attacker skills, etc. Another section lists the security principles affected by the attack. But, we observed that those given often have a high level of abstraction making their interpretation too abstract as well.

The section called “Attack Execution Flow” provides a sequence of steps that has to be followed to successfully accomplish an attack. The first step is often called “Explore”; it is often followed by the steps “Experiment” and “Exploit”, themselves composed of sub-steps. All of these are sequential, which means that if a step cannot be achieved then it is assumed that the attack cannot be applied. In other terms, it seems pointless to try the next step. Every step is accompanied with a sub-section called “Security controls”, which lays down some effective security controls that should be used to prevent or to counter the attack step. Furthermore, the CAPEC base provides some techniques (or combinations of techniques) with each step. When one technique is successfully applied, the step is satisfied.

1.2.3. Security principles

We briefly recall that 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 [32]. 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 proposed a set of eight best practices for system security [25], which were widely expanded to form security principles [32,26,7,14]. Most of the papers dealing with security principles reveal that a security principle has a level of abstraction; it may be the realisation of other security principles, or has subordinate principles.

2. Data-store

We present in this section the meta-model of the data-store we devised to store relationships among different security concepts. This data-store is then used to automatically generate a security pattern classification providing the set of patterns that can be used as countermeasures against a given attack (in reference to the security pattern definition of Schumacher [27]). Then, we present the data integration steps.

2.1. Data-store Meta-model

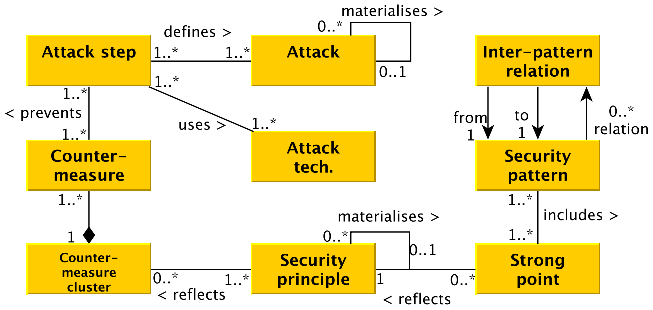


Figure 2. Metamodel 1 of the data-store

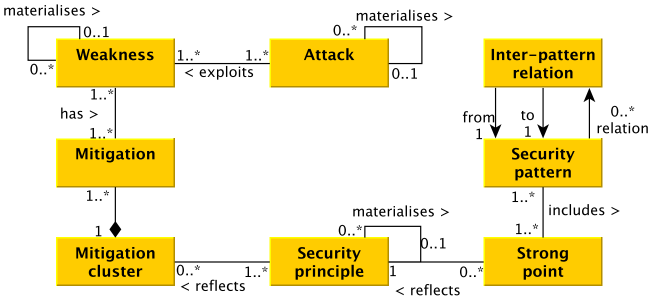


Figure 3. Metamodel 2 of the data-store

Instead of finding some direct relations among attack and security patterns by reading documents, we chose to decompose the security concepts available in these documents into more detailed properties, which can be interconnected in an explicit manner.

We surveyed the literature and some attack bases [15,16,19] to list relationships among security properties. This study has 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. All these properties and associations are modelled with entities in the meta-model of Figure 2. Taking another viewpoint, an attack also exploits a weakness, which may be composed of several more concrete weaknesses. Some actions can be applied to reduce the impact of these weaknesses. We call them mitigations here to underline the fact that these actions differ from the countermeasures used to neutralise an attack. These other associations are modelled in the meta-model of Figure 3.

Security patterns can be characterised with strong points, which are criteria of software engineering quality partially deduced from the section consequences in the pattern descriptions. In the context of security patterns, these correspond to desirable security properties. Besides, 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 a laborious task as these properties, which have diverse purposes, are explained with different keywords. To solve this issue, we have chosen to focus on security principles as mediators. Indeed, as introduced by Wassermann et al. in [33], security patterns are classifiable w.r.t. security principles. Here again, we consider that security principles are organised into a hierarchy, which shows the materialisation of a principle with more concrete ones. A principle hierarchy offers a lot of flexibility to reach the same abstraction level among strong points, principles, countermeasures and mitigations. Countermeasures and mitigations are often detailed security properties. We observed that gathering them into groups (clusters) often reduces the efforts required to find connections with security principles. But the cluster granularity, i.e., the size of the groups has to be correctly chosen not to set wrong associations. These last security properties and associations are identically modelled in the meta-models of Figures 2 and 3.

Both meta-models could be used to structure our data-store. We have chosen to focus on the first meta-model because it offers the possibility to store more details about attacks (decomposed into steps, techniques and related to countermeasures). A designer can follow how an attack is sequentially performed. As attack steps are associated to security principles and finally to security patterns, he or she also can obtain and select the list of patterns that are required to counter every attack step, one after the other. Hence, the relations among attacks, steps, principles and patterns offer a refinement in the pattern choice that the second meta-model does not provide.

Now that we have a data-store meta-model, the next section shows how the data integration is performed in order to extract a security pattern classification and ADTrees.

2.2. *Data integration and consolidation steps*

Security data are integrated into the data-store with six steps, which aim at establishing the different relations depicted in Figure 2. 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 others as expected. The steps 1 and 6 are automatically done with tools whereas the remaining ones require some manual interventions to supervise the digitalisation of key concepts or texts.

We implemented these steps and applied them to the Web application context as a proof of concept. The tools and databases are available in [21]. The implementation is mostly based upon the tool Talend, [2] an ELT (Extraction, Load, Transform) tool, which allows an automated processing of data independently from the type of its source or destination. We considered data coming from different sources: the CAPEC base, several papers dealing with security principles [25,32,26,7,14] and the pattern catalogue given in [38]. We provide some quantitative results related to these data with each step. But other kinds of context could be dealt with as long as enough documentation is available.

We also illustrate these steps with the attack "CAPEC-34: HTTP Response Splitting", which refers to a maliciously HTTP request that causes the production of two separate responses instead of one by a vulnerable web server. The target, i.e., the client, may interpret the second response and display maliciously-crafted contents.

2.2.1. *Step 1: Extraction of attacks, steps, techniques and security controls*

We have chosen to focus on the CAPEC base to extract information about security attacks because this appeared to be the most complete base with the largest number of attacks accompanied with a lot of details (steps, techniques, risks, security controls, etc.)

We extracted attacks of 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 "Related Attack Patterns". More precisely, by scrutinising all the CAPEC documents, it becomes possible to develop a hierarchical tree whose root node is unlabelled and connected to the attacks of the type "Category". These nodes may also be parent of attacks that belong to the type "Meta Attack pattern" and so on. The leaves are the most concrete attacks of the type "Detailed attack pattern". Then, for every attack, we collected from the CAPEC base (Section "Attack Execution Flow") its steps, which may be composed of more concrete sub-steps, and for each step, the corresponding techniques and security controls, the latter referring to countermeasures.

[2] https://talend.com/

This data extraction was automatically performed and yields a database DB_1. From the CAPEC database Version 2.8, we automatically extracted these elements for the Web application context and collected 215 attacks, 209 steps, 448 techniques and 217 countermeasures, knowing that attacks can share steps, attack techniques and countermeasures.

The attack CAPEC-34 has no sub-attacks as it belongs to the section "Detailed Attack" of the CAPEC base. The realisation of the attack is done after following three main steps called Explore, Experiment, Exploit. The third one is itself composed of two steps. Three techniques are listed to achieve the first step; the other steps are linked to two techniques each. Any of the available technique can be used to accomplish the related step.

2.2.2. Step 2: Countermeasure hierarchical clustering

The countermeasure number grows quickly while reading the attacks of the CAPEC base. Many of them have a close meaning though, which can be explained by the number of different contributors that added them. We hence group these countermeasures into families to later associate them with security principles.

We semi-automated this process by applying a hierarchical clustering technique of documents. We firstly used the tool KHcoder [3] to measure similarities among countermeasure descriptions. KHcoder is a free tool, referred by numerous works, which performs quantitative content analysis (text mining). We applied KHcoder as follows:

1. the tool POS Tagger (included in KHcoder) is called to sort the keywords found in the countermeasure descriptions (log, input, credentials, etc.) by their frequencies and types (noun, verb, adverb, etc.);
2. from the frequencies, weights are computed and scaled with the Jaccard coefficient to measure distances (a.k.a. dissimilarities) among countermeasures. The distance between two countermeasures a, b is defined by $0 \leq d_{a,b} = q+r/p+q+r \leq 1$, where p is the number of keywords occurring in a and b, q is the number of keywords occurring in a and not in b and r is the number keywords occurring in b and not in a. The more two countermeasure descriptions have common keywords, the more their distance is short. At the end of this step, we obtain a distance matrix containing distances between pairs of countermeasures.

Afterwards, we chose to apply the method Ward, an agglomerative hierarchical clustering method [35], to semi-automatically make a hierarchy of countermeasure clusters. Ward offers the possibility to merge groups, piece by piece, instead of providing large clusters. In our case, this second solution would tend to build clusters covering too much disparate countermeasures, which would be later associated with too much security principles. Instead, Ward successively constructs levels of clusters, a level somehow expressing a level of abstraction. Its algorithm is summarised in Algorithm 1. At the beginning, Ward takes a distance matrix, here previously computed by KHcoder. Every countermeasure is encompassed into a new cluster. The algorithm merges every pair of clusters having the closest distance into a new cluster and so forth. Every time a new cluster is created, the algorithm updates the distance matrix. The distance between two clusters is calculated with the formula $\triangle_{A,B} = n_A n_B / n_A + n_B \parallel \vec{m_A} - \vec{m_B} \parallel^2$, where n_A is the number of the elements of the cluster A and $\vec{m_A}$ is the gravity center of the cluster

[3] `http://khc.sourceforge.net/en/`

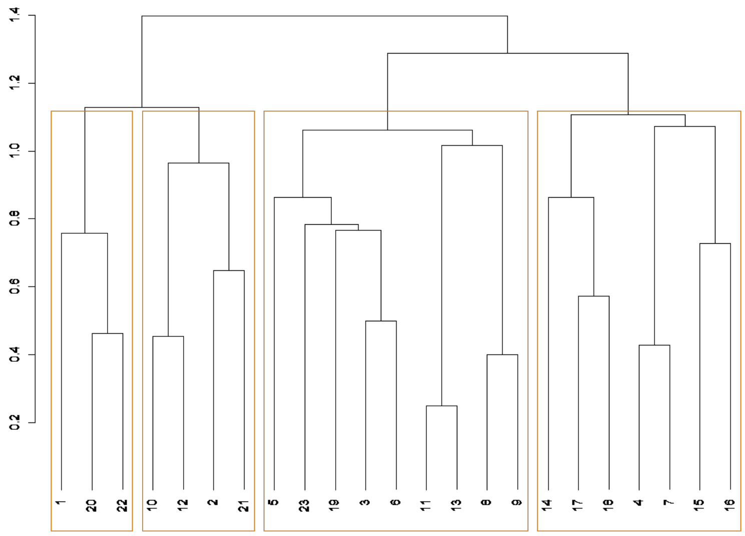


Figure 4. Hierarchical clustering of 23 countermeasures into 4 clusters using KH Coder

A, which represents the mean of the distances among the elements of the cluster A. At the end of the algorithm, all the clusters are grouped into one big cluster.

Algorithm 1: Hierarchical clustering

```
Require: distance matrix;
  repeat
      Find the closest pair of clusters;
      Merge them;
      Compute the gravity center of the new cluster;
      Update the distance matrix;
  until There is one cluster
```

Finally, the level to consider in the cluster organisation (and implicitly the number of clusters to keep) is selected manually, as the choice of the number of clusters is supervised in the domain of natural languages [29]. The level can be selected with a dendrogram. Figure 4 illustrates an example of dendrogram, obtained with 23 countermeasures. At the lowest level, the dendrogram shows all the countermeasures and its top level represents one final cluster. The choice of the number of clusters to keep comes down to draw an horizontal line in the dendrogram and to enumerate the number of cut vertical lines. There are two basic criteria to consider when inserting the line: a low cut is divisive, i.e., it may place two similar countermeasures in different clusters; a high cut is agglomerative, i.e., it may put in the same cluster two unrelated countermeasures. Therefore, in order to get a coherent clustering, the most suitable level has to be chosen after some iterations by checking whether the countermeasures obtained in the clusters refer to the same security principle or set of principles. In this example, we obtained four clusters.

The resulting clusters are stored into the database DB_2. The 217 countermeasures collected in Step 2, are aggregated into 21 clusters.

2.2.3. Step 3: Security patterns and strong points integration

We manually collected security patterns and their strong points from the catalogue given in [38]. Strong points are seldom explicitly provided, and have to be deduced from the pattern descriptions, more precisely from their forces and intents. Then, 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, which can be shared with other patterns;
2. the second relation defines inter-pattern connections based upon the annotations "depend", "benefit", "impair" or "alternative" [37]. With P a set of patterns, this relation is defined as a mapping from P^2 to the annotation set $\{"depend", "benefit", "impair", "alternative"\}$ that provides annotations for pairs of patterns.

These data and relations, which provide connections among security patterns and strong points, are encoded into the database DB_3. For the Web application domain, we have gathered 26 security patterns and 36 strong points.

2.2.4. Step 4: Security principle integration

We collected 66 security principles related to Web applications from the papers [25,32, 26,7,14]. Then, we organised them 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 counter an attack and provided by security patterns at the same time. As principles are hierarchically organised, we can link a strong point and a countermeasure cluster through this principle organisation 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 at two different levels of the hierarchy. As a principle is a child of the second one, it is possible to find an association between the strong point and the cluster.

The resulting hierarchy is certainly not exhaustive but covers the security patterns dealt with in the catalogue given in [38]. 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 the most abstract principles, e.g., “Access control”, and the lower level exhibiting the most concrete principles, e.g., “File authorization”.

2.2.5. Step 5: Associations among strong points, security principles and countermeasure clusters

In this step, we established the many-to-many relation between strong points and security principles. We have chosen to manually integrate this relation because strong points and principles are mostly presented in an abstract manner, with textual documents. We observed that the abstraction level of the strong points better fits with the most concrete principles, which are the leaves of the hierarchical organisation depicted in Figure 5.

Afterwards, we established the many-to-many relation between countermeasures clusters and security principles. After Step 3, the clusters include countermeasures sharing the same security concepts. Once these concepts are known, linking clusters to se-

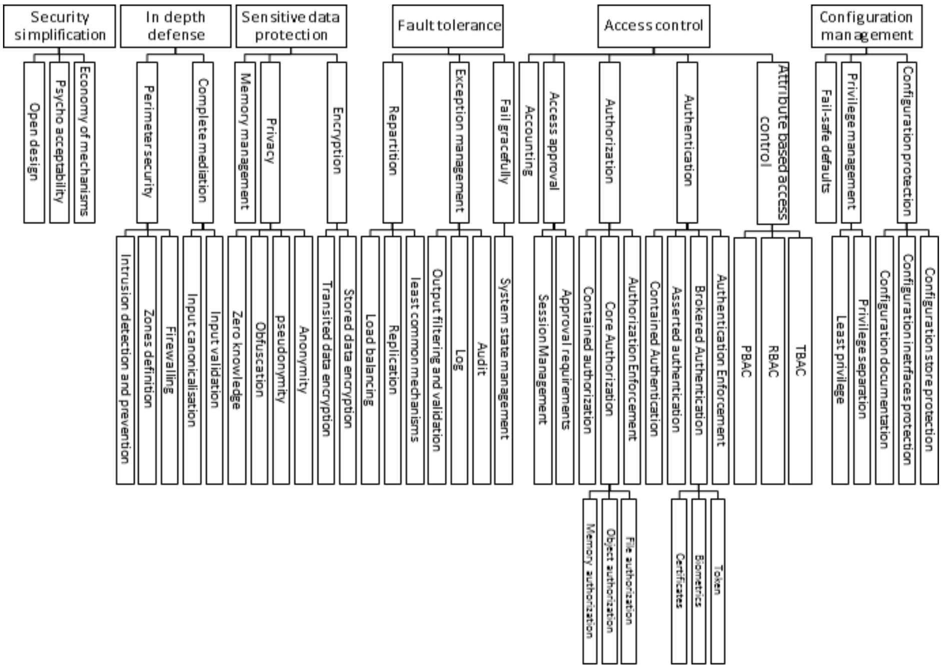


Figure 5. Hierarchical organisation of security principles

curity principles becomes straightforward, as principles are often defined with regard to these same concepts.

These relations are materialised with the database DB_5, which combines 21 clusters, 36 strong points and 66 principles.

If we take back our example of attack CAPEC-34, its first step “Explore” aims to explore a Web application to record its user-controllable input points. A countermeasure of this attack step consists in storing and auditing all the application accesses to detect the application exploration. Only the administrator should be able to perform this task. This countermeasure belongs to a cluster that is associated to the principles “Audit”, “Log” and “File Authorization”. We associated “Log” with the strong point “log sensitive information that should not be accessible to unauthorised users”, which finally belongs to the security pattern “Secure Logger”.

2.2.6. Step 6: data consolidation

The previous databases DB_1 to DB_5 are now combined into a single one. On the one hand, DB_1, DB_2 and DB_5 store the relations among attacks, steps, countermeasures and principles. On the other hand, DB_3 and DB_5 store the relations among security patterns, strong points and principles. It is now manifest that the security principle hierarchy becomes the central point that helps map attacks onto security patterns.

In our implementation, this step is automatically performed by the tool Talend by means of the meta-model given in Figure 2. As two databases do not share more than one entity of the meta-model, this process does not raise any particular issue. This step produces the final database DB_f, which is available in [21].

3. Security Pattern Classification and ADTree Generation

The final database DB_f holds enough information to classify security patterns and generate ADTrees. We present below how to automatically generate them.

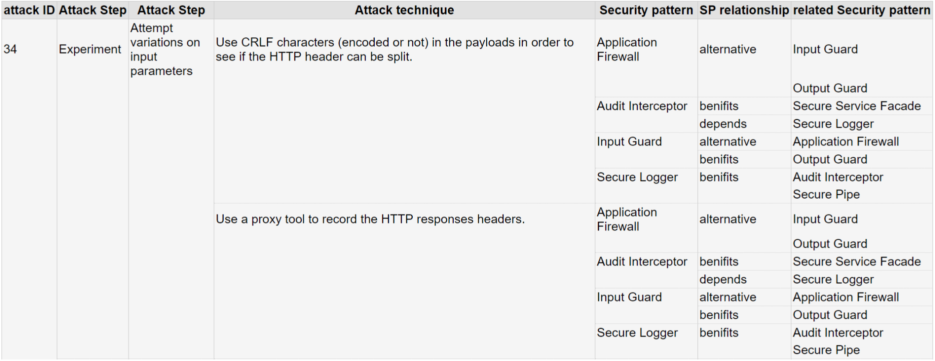
3.1. Security pattern classification

By means of the relations defined in the meta-model of Figure 2, we extract from DB_f a catalogue listing the combinations of security patterns that are countermeasures against an attack. We derived, from the meta-model of Figure 2, a graph taking back the relations defined from the entity Attack up to the entity Security pattern. The data extraction can be automatically performed from this graph with a tool like Talend. Given an attack Att, the following data and relations are extracted from DB_f:

- the information about the attack (name, identifier, description);
- the tree $T(Att)$, whose root is Att, if Att is not a leaf of the tree derived in Step 1. For every attack found in $T(Att)$, we also extract its attack steps and techniques;
- for each step st, the complete hierarchy of security principles $Sp(st)$ by means of the successive relations established among st, countermeasure clusters and security principles. $Sp(st)$ denotes the complete hierarchy of security principles related to a step, i.e., if a principle sp of $Sp(st)$ is not a leaf of the hierarchical organisation depicted in Figure 5, then we also extract all the principle sub-tree whose root is sp. As the CAPEC base does not clarify whether all the countermeasures or only some of them have to be used for preventing an attack step, we have chosen to take them all into consideration and all the respective security principles;
- for each principle sp in $Sp(st)$, 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}. In addition, we extract the inter-pattern relations defined for couples of patterns by the annotations in $\{"depend", "benefit", "impair", "alternative", "conflict"\}$.

After the data extraction, we obtain a security pattern classification presented in a tabular form. The data integration steps and the classification extraction offer the advantage of semi-automatically achieving a security pattern classification that may be updated. For instance, if one want to add a new attack, the steps 1, 2 and 5 have to be followed. Likewise, if a new security pattern is proposed in the literature, the steps 3, 4 and 5 have to be applied. And the classification extraction can be re-executed every time the data-store is updated. From our database DB_f given in [21], we have automatically extracted a security pattern classification specialised to the Web application domain, which includes 215 CAPEC attacks and 26 security patterns.

Figure 6 depicts an extraction example for the attack CAPEC-34. The first column gives the attack ID. This attack has no sub attacks (otherwise, the next columns would list them too). Columns 2 to 4 index the attack steps and techniques. To ease readability, we only illustrate the step Experiment here. The security patterns allowing to prevent the step are given in Column 5. These four patterns have to be integrated in the application model and implemented to prevent the attack. The last two columns list the security patterns being associated with the patterns of Column 5 and their relations. For instance, Figure 6 reveals that "Application Firewall" and "Input guard" are alternative patterns, hence using one of them is enough (although using both is not incorrect). Figure 6 also



attack ID	Attack Step	Attack Step	Attack technique	Security pattern	SP relationship	related Security pattern
34	Experiment	Attempt variations on input parameters	Use CRLF characters (encoded or not) in the payloads in order to see if the HTTP header can be split.	Application Firewall	alternative	Input Guard
						Output Guard
				Audit Interceptor	benifits	Secure Service Facade
					depends	Secure Logger
				Input Guard	alternative	Application Firewall
					benifits	Output Guard
				Secure Logger	benifits	Audit Interceptor
						Secure Pipe
			Use a proxy tool to record the HTTP responses headers.	Application Firewall	alternative	Input Guard
						Output Guard
				Audit Interceptor	benifits	Secure Service Facade
					depends	Secure Logger
				Input Guard	alternative	Application Firewall
					benifits	Output Guard
				Secure Logger	benifits	Audit Interceptor
						Secure Pipe

Figure 6. Extraction of the pattern classification for the attack CAPEC-34

illustrates that "Secure Logger" may benefit from the security patterns "Secure Pipe" or "Audit Interceptor".

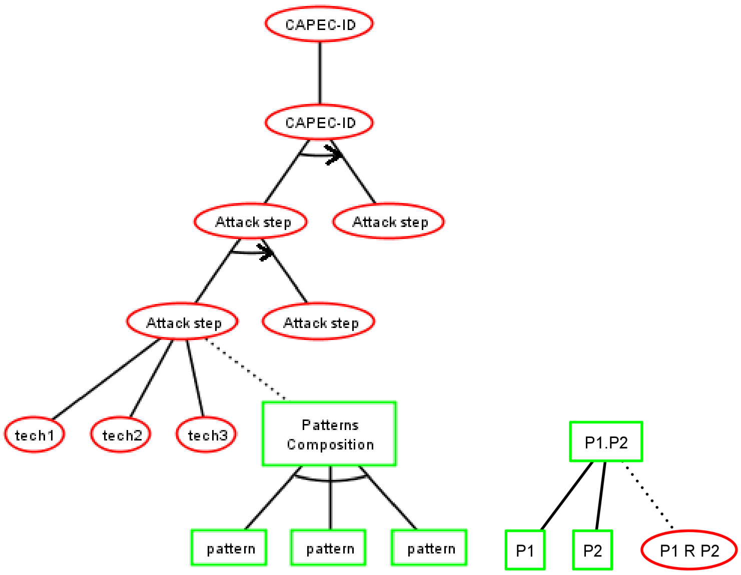
A designer can interpret this extraction to select security patterns in a precise manner step after step. For instance, the attack step Experiment refers here to the sending of malicious requests by means of the application entry points (URLs, forms, etc.). These entry points were identified by the attack step Explore earlier. The first security solution is to validate requests either with the security pattern "Input Guard" or "Application Firewall". The choice of the pattern mostly depends on the application design and features. "Application Firewall" allows the decoupling of the input validation from the remainder of the application. But it is also more cumbersome to implement than "Input Guard" as it aims at filtering all the application requests and responses.

Figure 6 reveals that our classification does not list all the data available in the data-store and related to an attack, e.g., countermeasures or strong points. We stated in Section 2 that this information is mostly used to establish direct relations among security concepts to eventually generate links between attacks and security patterns. We have chosen to generate a classification that includes these links but not all the underlying details to make it more readable. However, the data-store can still be queried to extract more related details about attacks (techniques, counter-measures, affected principles) or patterns (strong points, principles).

Raw tabulars may not be easily comprehensible for beginners in security or in patterns. Actually, this classification representation may contradict the criterion Comprehensibility , which refers to the ability of using the classification by experts or novices. This is why we supplement the classification with graphical models called ADTrees to improve readability.

3.2. Attack-Defense Tree generation

Attack Defense Trees *"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"* [13]. We recall that ADTrees have two different kinds of nodes: attack nodes (red circles) and defense nodes (green squares). A node can be *refined* with child nodes with disjunctive or conjunctive refinements. The former is recognisable by edges going from a node to its children. The latter is graphically distinguishable by connecting



(a) Pattern classification representation with ADTrees

(b) Conflicting pattern representation with ADTrees

Figure 7. Generated ADTree forms

these edges with an arc. Here, we extend these two refinements with the sequential conjunctive refinement of attack nodes, defined by the same authors in [11]. This operator expresses the execution order of child attack nodes. Graphically, a sequential conjunctive refinement is depicted by connecting the edges going from a node to its children with an arrow.

We generate ADTrees having the general form illustrated in Figure 7(a). An ADTree root node is labelled by an attack. This root node may be disjunctively refined with sub-attacks. When an attack is defined with steps, its node is refined with child nodes labelled by these steps (sequential conjunctive refinement). The most concrete steps are graphically represented with attack nodes refined with other attack nodes labelled by techniques (disjunctive refinement). A node labelled by an attack step has one child defense node (in green in Figure 7(a)), which may be the root of a defense sub-tree expressing security pattern combinations.

ADTrees are obtained with the following steps:

1. a new ADTree is generated for every attack stored into DB_f. Its root node is labelled by the attack 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;
2. for each attack Att found in the preceding tree, we collect its sequence of steps. The node labelled by Att is refined with a sequential conjunction of attack nodes, one for each step. We repeat this process if a step is itself composed of steps. In the same way, for each step St, the related techniques are extracted from the datastore and are associated to the node labelled by St with a disjunctive refinement;

3. for each step st, we extract the set P of security patterns that counter st. Given a couple of patterns $(p_1, p_2) \in P$, we illustrate their associations with new nodes and logic operations as follows. 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 $p_1 \; R \, p_2$ 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 $p_1 \; alternative \, p_2$ 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 $\{impair, conflict\}$. The operator XOR sounds to be the evident candidate for modelling this case. Unfortunately, this operator is not defined in this tree model. Therefore, we use the classical formula $(p_1 \; xor \; p_2) \longrightarrow ((p_1 \; or \; p_2) \; and \; not \; (p_1 \; and \; p_2))$, except that the not operator (unavailable with ADTrees) is here replaced by an attack node labelled by $(p_1 \; R \, p_2)$ meaning that two conflicting security patterns used together constitute a kind of attack. The corresponding sub-tree 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 root defense node is linked to the attack node labelled by st. We consider all the security patterns here (combined with a conjunctive operator) on account of the lack of clarification of the CAPEC base. Indeed, as stated in the classification generation, the CAPEC base does not report whether all the countermeasures or only some of them have to be used for preventing an attack step. Hence, we have chosen to consider them all and all the related security patterns.

When an attack step is linked to several security patterns, the second step may achieve a large defense sub-tree. This one can be reduced though by using logical expression simplifications. 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 classical logical expressions. These 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 to (p1 AND p2). This expression gives a defense node that is conjunctively refined with two nodes labelled by p1 and p2.

An ADTree resulting from the previous steps represents all the possible scenarios that can lead to the realisation of the attack given in the root node. It provides sequences of attack steps and techniques that have to be executed in the right order to perform the attack with success. On the opposite side, it also includes defense nodes, which may be the roots of sub-trees expressing combinations of security patterns. It remains for the designer to chose one combination of patterns for every step at the application design stage.

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

We implemented the ADTree generation with a tool available in [21]. It takes as input an attack identifier and yields an ADTree, which is stored into an XML file. ADTree files can be modified or updated as the designer wishes with the tool given in [12].

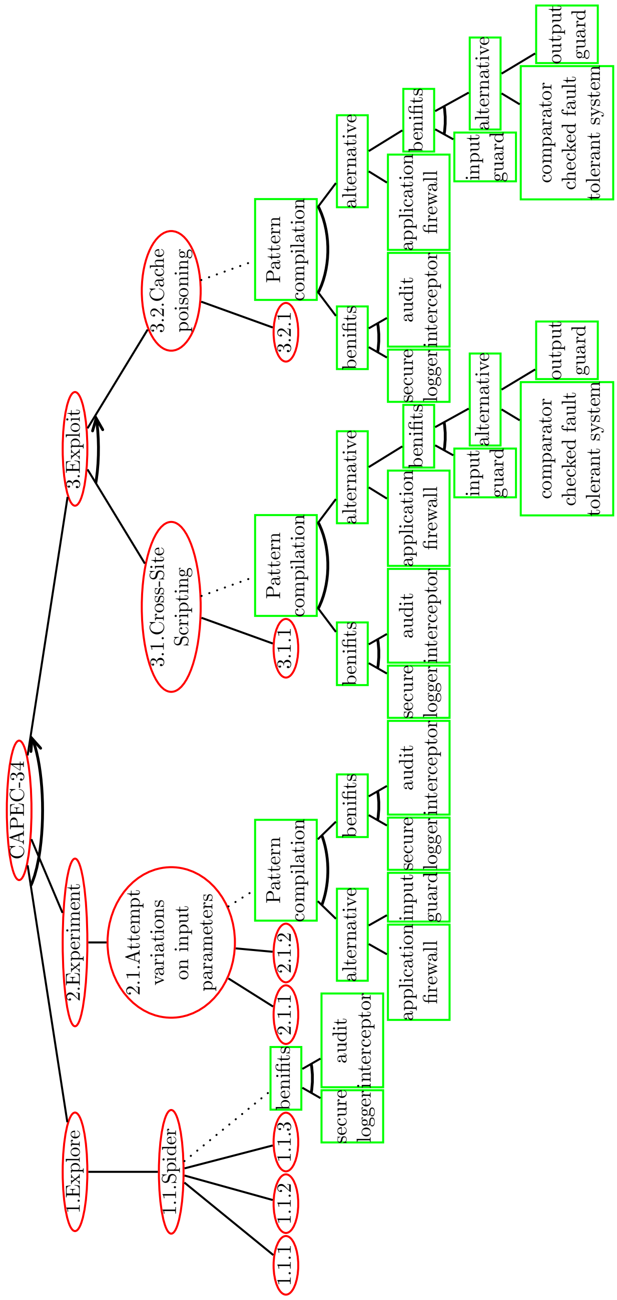


Figure 8. ADtree of the Attack CAPEC-34

Table 1. attack techniques descriptions

TECHNIQUE	TECHNIQUE DESCRIPTION
1.1.1	Use a spidering tool to follow and record all links and analyze the web pages to find entry points. Make special note of any links that include parameters in the URL, forms found in the pages (like file upload, etc.).
1.1.2	Use a proxy tool to record all links visited during a manual traversal of the web application.
1.1.3	Use a browser to manually explore the website and analyze how it is constructed. Many browsers' plugins are available to facilitate the analysis or automate the discovery.
2.1.1	Use CR/LF characters (encoded or not) in the payloads in order to see if the HTTP header can be split.
2.1.2	Use a proxy tool to record the HTTP responses headers.
3.1.1	Inject cross-site scripting payload preceded by response splitting syntax (CR/LF) into user-controllable input identified as vulnerable in the Experiment Phase.
3.2.1	The attacker decides to target the cache server by forging new responses. The server will then cache the second request and response.

Figure 8 illustrates the ADTree obtained for the attack CAPEC-34 and Table 1 lists the techniques labelled in this ADTree. The root of the tree is the main goal of the attacker. Its second and third levels relate to the attack steps. These nodes are sequential conjunctive refinements of the root node. For instance, the step Exploit is achieved if both steps 3.1 and 3.2 are successfully executed in the right order (from left to right). An attack step has a disjunctive refinement of attack nodes labelled by techniques. The step is achieved if one of the attack techniques is applied with success. The lower nodes, labelled by attack steps, are linked to (green square) defense nodes, which illustrate security pattern combinations. We observe that the step 1.1 "Spider" can be prevented by designing the application with both patterns "Audit interceptor" and "Secure logger". "Audit interceptor" can be used to detect the application crawling and to warn an administrator. The audit logs are secured by means of "Secure logger", which guarantees that the audit logs cannot be accessed or altered by unauthorised users.

4. Classification Discussion

We discuss in this section the quality, accuracy and the limitations of our classification.

4.1. Classification Quality

To assess the quality of this classification we studied the nine criteria proposed by Alvi et al. in [2]. Our classification meets seven of these criteria:

- Navigability: our classification (supplemented with ADTrees), satisfies this criterion as it exposes the hierarchical refinements of an attack and, for every attack step, the combinations of patterns, which should be integrated in the application model. Besides, the classification provides the relationships among security patterns, which help choose the most appropriate pattern combination;
- Determinism: the classification is clearly defined by means of the methodology steps. All these steps justify the soundness of the classification;

- Unambiguity/Comprehensibility: as patterns are classified w.r.t. attacks, steps, and security principles, we provide a clear structure of categories. This organisation, which is 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 a known security pattern catalogue [38] and upon the CAPEC base, which is more and more employed in the industry. 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 [12]. 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 and security principles. Furthermore, the evaluation given in Section 5 suggests 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.

Our classification does not yet satisfy two quality criteria called Mutual exclusivity (patterns should be categorised into at most one category) and Completeness (all the existing security patterns are covered). Mutual exclusivity does not hold because a security pattern can be related to several attacks and security principles in the meta-model of Figure 2. Even though this is not a primary goal of our classification, we could fix this issue by grouping the most concrete attacks into contexts in a mutual way as in [5]. To do so, the meta-model of Figure 2 should be updated with a new entity called Context linked to an entity Concrete Attack itself linked to the Entity Attack.

4.2. Classification Accuracy

We conducted a systematic review of the data integration steps and of the classification to ensure that the security patterns provided to counter an attack are effective and that none is missing among the 26 patters considered in this paper. Unfortunately, this review had to be manually done as most of the security concepts considered in the meta-model of Figure 2 are abstract in nature. In short, the review process was carried out as follows. The first author studied the associations between security patterns and principles (steps 3, 4 and 5). Some relations were corrected and strong points were added during this process. The second author audited the associations between attacks and principles (steps 2 and 5). In particular, the clusters of counter-measures were carefully examined as the clustering technique requires supervision. Step 1, which involves the automatic extraction of information about attacks was quickly examined as the extraction was done from a public base regularly reviewed by thousands of users. Then, we asked two master students to review half of the pattern classification each one, i.e. the patterns provided to counter every attack. One of the authors reviewed the complete classification. The other author studied the associations between patterns and security principles and especially checked whether some associations between strong points and principles were missing. When there was a disagreement in answers, we discussed the issues until we reached an agreement.

Furthermore, to validate the accuracy of the classification, we also compared it with the results issued in the two papers dealing with the associations between patterns and attacks/weaknesses [34,1]. 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. The relations exposed in [34,1] does not reveal any inconsistency with our classification. For instance, the attack "CAPEC-66 SQL Injection" is related to the security patterns "Intercepting Validator" and "Input validation" in [34]. 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. These ADTrees give equivalent patterns. For instance, the ADTrees exhibit the pattern "Input Guard", which can be implemented by "Intercepting Validator". But, they also list other security 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 countermeasures of 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. More generally, we have observed that our classification exposes more pattern combinations per attack; the more choice is not always the better though. But, after inspection, we have concluded that more than one or two patterns are generally required to counter attacks.

4.3. Limitations

After the review of our classification, we have observed that it presents some limitations, which could lead to some research future work.

The notion of attack combination is not considered in the paper. 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, the resulting ADTree size may 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) and 26 security patterns out of 176. It can be completed with new attacks automatically. But the completion of the data-store with new security patterns requires some manual steps. It could be interesting to investigate whether text mining techniques would help partially automate them. The classification exhaustiveness also depends on the available security data. In the ADTree of Figure 8, all the lowest attack nodes are linked to defense nodes. We sometimes observed that no defenses are provided with other attacks. This can be usually explained by three main reasons:

1. security databases or pattern catalogues are incomplete (lack of mitigation, countermeasure, etc.). More data are required during the data integration process. In particular, we observed that some countermeasures are missing for some attacks of the CAPEC base;

2. the attack is relatively new. It is not documented yet or no pattern based solution is available;
3. security data are missing because we did not considered them in the manual data integration steps. For instance, as the pattern descriptions do not clearly provide strong points, it may be easy to skip one of them.

Several steps require manual interventions, which are prone to errors. These steps may lead to associations among security data that are bound to be controversial. We reviewed and compared our classification with other papers to check whether the selected patterns are appropriate. As these papers provide few associations between patterns and attacks, our validation process is incomplete. Validating every relation of the meta-model of Figure 2 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 another long and error-prone task that should be addressed.

Finally, the inter-pattern associations are defined with binary relations only, as presented in [37]. These relations could be updated to link several patterns together.

5. Empirical Evaluation

We evaluated our classification to ensure that the previous quality criteria can be met in practice. We empirically studied two scenarios where 24 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 following a block release training; most of them have good skills in the design, development and test of Web applications (PHP, Javascript). 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: some UML sequence diagrams capturing the main functionalities of the application, the CAPEC base, two concrete examples showing how to perform each attack, the catalogue of security patterns given in [38] and the pattern classification proposed in [1]. The catalogue includes 36 patterns whose most of them can be used with Web applications. For simplicity, we refer to these documents as basic 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 giving the attack steps, techniques and combinations of security patterns (as in Figure 6), two ADTrees generated from the data-store (Figure 8 is one of them). 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)?

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

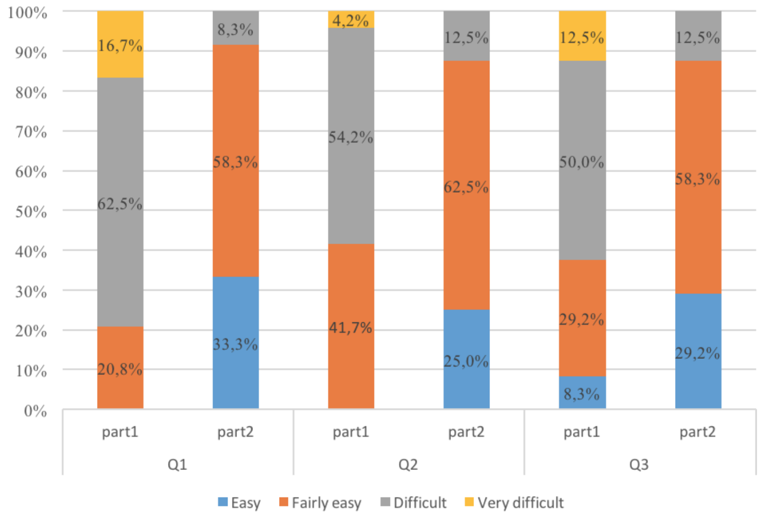


Figure 9. Response rates for Q1 to Q3

- 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 devised to evaluate these three 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 ?

5.1. Experiment results

From the forms returned by the participants (available in [21]), we extracted the following results. Firstly, Figure 9 illustrates the percentages of answers to the questions Q1 to Q3. For these, we proposed 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. We gauged 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*. The bar charts of Figure 10 depicts the levels of confidence of the participants.

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: we considered that any pattern combination allowing to counter the attack is a good solution. Several pattern combinations were accurate. When a participant gives one of these combinations, its response belongs to this category;

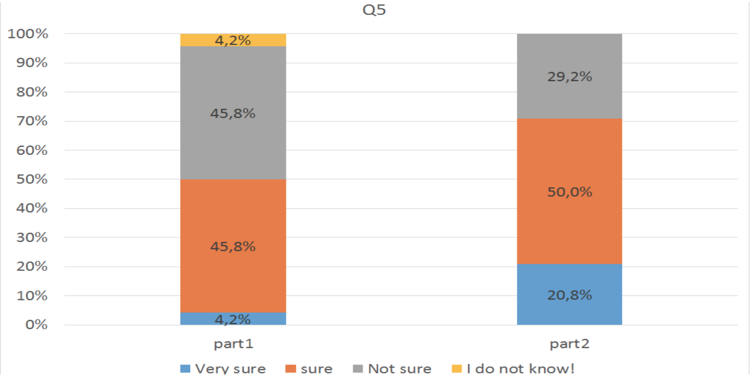


Figure 10. Confidence rates (Q5)

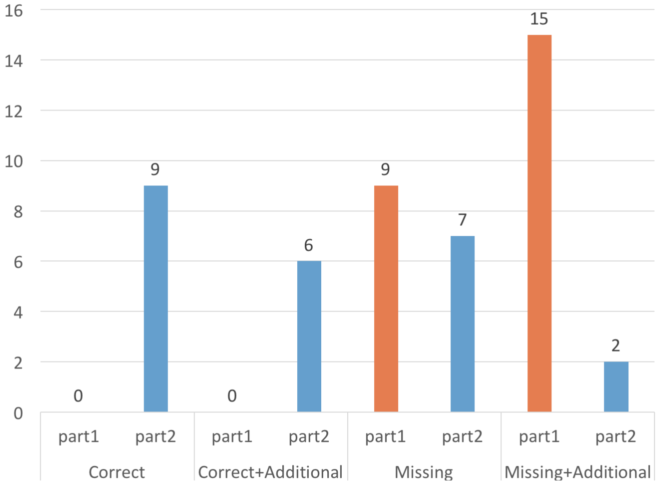


Figure 11. Accuracy Measurement (Q6)

- Correct+Additional: this category includes the responses composed of a correct pattern combination, completed with some other unnecessary patterns;
- Missing: we gather in this category, the incomplete pattern combinations without additional patterns;
- Missing+Additional: this category holds the worst responses, composed of unnecessary patterns eventually accompanied with some expected ones but not of of them).

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

5.2. Result interpretation

5.2.1. 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. In addition, Figure 10 expresses that the confidence of the participants on their responses increased by 20,8 %.

5.2.2. 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.

5.2.3. 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 24 (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.

5.3. Threat to Validity

There are many application and system contexts, but this preliminary experimental evaluation is applied on Web applications only. This is a threat to external validity, in the sense that the results about Comprehensibility and Accuracy cannot be generalised to all software systems. This is why the experiments deliberately avoid drawing any general conclusion. But, this threat is somewhat mitigated by the considered context itself. The Web development is indeed a rich field in great demand in the software industry. Web applications also expose a lot of well-known vulnerabilities. Besides, this well-studied application context helped us propose experimentations involving participants having the adequate knowledge on software development and security.

This leads to the second threat to validity concerning the audience. Our evaluation was indeed performed on a public of students following a block release training. This sort of subject is sometimes considered as a bias, as any strict process should help them improve their work. But, several studies conclude that student experiments are appropriate as evaluation for software engineering approaches, especially when Comprehensibility is a criterion taken under evaluation [6]. We also believe that we would have achieved the same results with a group of developers from the industry, as they often do not have any software security skills. Evaluating our tools with a public of security experts could be interesting, but this work initially do not target this kind of audience.

Another threat relates to the learning effect, which may happen between the two stages of the evaluation: we indeed applied the same approach to the same participants. We only replaced the Web application in Part 2 (to avoid another bias, we took care to provide an application providing similar functionalities and exposing the same vulnerabilities), and we completed the available security documents by a list of ADTrees. We deliberately chose to apply the same approach in Part 1 and 2 to evaluate the felling of the participants about the security documents (Q3, Q2) and their confidence on their work (Q5). In other words, the fact of changing the audience might have brought a bias on our analysis of Comprehensibility. But, it is indisputable that this kind of experiment may influence our results. We tried to mitigated this threat by changing the question order. Besides, our results suggest that there are significant improvements in Comprehensibility and Accuracy between Part 1 and 2. We believe that these improvements cannot only be explained by a learning effect and that our approach has a real impact here.

There is also a risk as we asked the participants to select security patterns independently of the application design, even though the latter was partly given to them. Hence, our results on Accuracy (depicted in Figure 11) might include patterns whose integration in the application is not possible. We studied the problem of pattern selection with regard to an existing model in another work [23]. We have shown that the strict analysis of pattern integration in a model is not trivial and depends on several factors, e.g., structural or behavioural integration. We thus believe that this study requires its own kind of experiment. In addition, the task of selecting patterns, which can fit in an existing model, would require much longer experiment times that those we considered. But, let us examine the impact of considering as correct only the patterns that fit in the application design. With regard to Figure 11, this new viewpoint would only affect the results of the two first columns: we should have less pattern combinations in the column Correct, which are transferred in the second column Correct + Additional pattern. The two last columns (Missing, Additional patterns) remain unmodified. This shows that our approach still increases Accuracy in this case.

We thus believe that the empirical experience reported in this paper provides relevant insights on the benefits of using our pattern classification.

6. Conclusion

We have proposed a security pattern classification method based on the integration of various security data. The method, composed of 6 manual and automatic steps, generates a data-store and a classification associating attacks, security principles and security patterns in order to help designers choose security pattern combinations to design secure applications. The method also builds ADTrees, which graphically illustrate the classification. As a proof of concept, we implemented these steps and generated a security pattern classification, which includes 215 CAPEC attacks, 66 security principles and 26 security patterns. We evaluated the quality of this classification by means of the criteria exposed in [2] and with an experimentation performed on 24 participants. The experimentation suggests that our classification and ADTrees make the pattern choice easier, more accurate, and the participants more efficient.

We also mentioned several limitations, which could lead to 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 could 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, which might impede Comprehensibility. The ADTree reduction could be a first solution to 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. Another line of future work is to integrate other security concepts in our datastore to provide other kinds of classifications. For example, several researchers studied the relations between security patterns and tactics. Tactics are described as *measures or decisions taken to improve quality factors* [4,9]. Patterns actually represent some realisations of tactics. In the meta-model of Figure 2, the notion of security tactic should find its place between principles and strong points. But, another way of integration would be to replace security principles with tactics. This alternative could refine the pattern selection, as the desirable security property might be more precise. But we estimate that the effort required to associate counter-measures, strong points and tactics would be substantial as there are many more tactics than principles. Once more, text mining might help reduce this effort. We will also continue to investigate the use of data integration techniques to simplify some steps of the software life cycle. In particular, we will study whether our generated ADTrees could serve for the test case generation.

References

[1] 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*, pages 113–120, 2011.

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

[3] 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*, pages 476–483, 2014.

[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

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

[6] M. Daun, C. Hübscher, and T. Weyer. Controlled experiments with student participants in software engineering: Preliminary results from a systematic mapping study. *CoRR*, abs/1708.04662, 2017.

[7] V. Dialani, S. Miles, L. Moreau, D. De Roure, and M. Luck. Transparent fault tolerance for web services based architectures. In *Euro-Par 2002 Parallel Processing*, pages 889–898. Springer, 2002.

[8] E. B. Fernandez. Security patterns and secure systems design. In A. Bondavalli, F. Brasileiro, and S. Rajsbaum, editors, *Dependable Computing*, pages 233–234, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[9] E. B. Fernandez, H. Astudillo, and G. Pedraza-García. Revisiting architectural tactics for security. In D. Weyns, R. Mirandola, and I. Crnkovic, editors, *Software Architecture*, pages 55–69, Cham, 2015. Springer International Publishing.

[10] D. Harb, C. Bouhours, and H. Leblanc. *Using an Ontology to Suggest Software Design Patterns Integration*, pages 318–331. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[11] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua. Attack trees with sequential conjunction. In *IFIP International Information Security Conference*, pages 339–353. Springer, 2015.

[12] B. Kordy, P. Kordy, S. Mauw, and P. Schweitzer. ADTool: security analysis with attack–defense trees. In *International Conference on Quantitative Evaluation of Systems*, pages 173–176. Springer, 2013.

[13] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, page exs029, 2012.

[14] J. Meier. Web application security engineering. *Security & Privacy, IEEE*, 4(4):16–24, 2006.

[15] Mitre corporation. Common attack pattern enumeration and classification, url:https://capec.mitre.org/, 2018.

[16] Mitre corporation. Common weakness enumeration, url:https://cwe.mitre.org/, 2018.

[17] H. Munawar. Security pattern catalog, http://www.munawarhafiz.com/securitypatterncatalog/.

[18] Offensive Security. Exploit database archive, url:https://capec.mitre.org/, 2018.

[19] OWASP. Owasp testing guide v3.0 project. In *http://www.owasp.org/index.php/Category:OWASP_Testing_Project#OWASP_Testing_Guide_v3*, 2003.

[20] L. Regainia and S. Salva. A methodology of security pattern classification and of attack-defense tree generation. In O. Camp, S. Furnell, and P. Mori, editors, *Proceedings of the 3nd International Conference on Information Systems Security and Privacy (ICISSP 2017*, Porto, Portugal, 02 2017. SciTePress.

[21] L. Regainia and S. Salva. Security pattern classification, companion site, http://regainia.com/research/companion.html, 2018.

[22] L. Regainia, S. Salva, and C. Bouhours. A classification methodology for security patterns to help fix software weaknesses. In *13th ACS/IEEE International Conference on Computer Systems and Applications AICCSA 2016*, Agadir, Morocco, 11 2016. IEEE Computer Society.

[23] L. Reigaigna, C. Bouhours, and S. Salva. A systematic approach to assist designers in security pattern integration. In *The Second International Conference on Advances and Trends in Software Engineering (SOFTENG 2016)*, Lisbon, Portugal, 02 2016.

[24] E. Rodriguez. Security Design Patterns. In *19th Annual Computer Security Application Conference (ACSAC'03)*, 2003.

[25] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

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

[27] M. Schumacher. *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[28] R. Slavin and J. Niu. Security patterns repository, http://sefm.cs.utsa.edu/repository/, 2017.

[29] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[30] I. A. Tøndel, J. Jensen, and L. Røstad. Combining misuse cases with attack trees and security activity models. In *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*, pages 438–445. IEEE, 2010.

[31] A. V. Uzunov and E. B. Fernandez. An extensible pattern-based library and taxonomy of security threats for distributed systems. *Computer Standards & Interfaces*, 36(4):734–747, 2014.

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

[33] R. Wassermann and B. H. Cheng. Security patterns. In *Michigan State University, PLoP Conf*. Citeseer, 2003.

[34] A. Wiesauer and J. Sametinger. A security design pattern taxonomy based on attack patterns. In *International Joint Conference on e-Business and Telecommunications*, pages 387–394, 2009.

[35] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[36] J. Yoder, J. Yoder, J. Barcalow, and J. Barcalow. Architectural patterns for enabling application security. *Proceedings of PLoP 1997*, 51:31, 1998.

[37] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen. A system of security patterns, technical report cw-469, 2006.

[38] K. Yskout, R. Scandariato, and W. Joosen. Do security patterns really help designers? In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 292–302, Piscataway, NJ, USA, 2015. IEEE Press.