A Methodology of Security Pattern Classification and of Attack-Defense Tree Generation

Loukmen Regainia and Sébastien Salva
LIMOS - UMR CNRS 6158, Auvergne University, Aubière, France
{*loukmen.regainia, sebastien.salva*}*@udamail.fr*

Keywords: Security Patterns, Classification, CAPEC Attacks, CWE Weaknesses, Attack-Defense Trees.

Abstract: Security at the design stage of the software life cycle can be performed by means of security patterns, which are viable and reusable solutions to regular security problems. Their generic nature and growing number make their choice difficult though, even for experts in system design. To guide them through the appropriate choice of patterns, we present a methodology of security pattern classification and the classification itself, which exposes relationships among attacks, weaknesses and security patterns. Given an attack of the CAPEC (Common Attack Patterns Enumeration and Classification) database , the classification expresses the security pattern combinations that overcome the attack. The methodology, which generates the classification is composed of five steps, which decompose patterns and attacks into sets of more precise sub-properties that are associated. These steps provide the justifications of the classification and can be followed again to upgrade it. From the classification, we also generate Attack-Defense Trees (ADTtrees), which depict an attack, its sub-attacks and the related defenses in the form of security pattern combinations. Without loss of generality, this classification has been established for Web applications and covers 215 attacks, 136 software weaknesses and 26 security patterns.

1 INTRODUCTION

Design patterns are recurrent 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 were also proposed to improve the security properties of models.

Security patterns are reusable (design) elements to design secure applications which will enable software architects and designers to produce a system which meets their security requirements and which is maintainable and extensible from the smallest to the largest systems (Rodriguez, 2003).

Security patterns relates countermeasures (stated in the solution) to threats and attacks in a given context (Schumacher and Roedig, 2001).

Hence, these patterns are often presented with UML (Unified Modelling Language) class and sequence diagrams.

Since 1997, the number of security patterns is continuously growing and around 170 security patterns are available at the moment (Slavin and Niu, 2016). Security patterns are often presented with a high level of abstraction to be reusable in different kinds of context and are described with textual documents. Because of this abstract nature and because the documents are not structured in the same manner, the choice of the most appropriate security pattern to mitigate a security problem is sensitive and somehow perilous to novice designers (Yskout et al., 2012; Alvi and Zulkernine, 2011). As designers cannot be experts in all the fields of software engineering, they clearly lack of guidance in the choice of patterns during the design phase. As a consequence, security pattern taxonomies were proposed in the literature to lead them towards good practices.

Several classifications were proposed to arrange security patterns in different kinds of categories, e.g., by security principles (Yskout et al., 2006; Alvi and Zulkernine, 2012), by application domains (Bunke et al., 2012) (software, network, user, etc.)., by vulnerabilities (Anand et al., 2014; Alvi and Zulkernine, 2011) or by attacks (Wiesauer and Sametinger, 2009; Alvi and Zulkernine, 2011). Despite the improvements in the pattern choice brought by these classifications, several issues still remain open. Among

Regainia, L. and Salva, S.
A Methodology of Security Pattern Classification and of Attack-Defense Tree Generation.
DOI: 10.5220/0006198301360146
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 136-146
ISBN: 978-989-758-209-7

them, we noticed that these classifications are all manually devised by directly comparing the textual descriptions of patterns with those of the other security artifacts (vulnerabilities, attacks, etc.). As these descriptions are generic and have different levels of abstraction, the categorisation of a pattern can be done only when there is an evident relation between it and a security property. Furthermore, it is often delicate to upgrade these classifications since no strict process to follow is available. Finally, the navigability among a set of patterns is rarely taken into consideration by theses classifications (Alvi and Zulkernine, 2012). This important quality criterion is defined as the ability to guide the choice of designers among collaborative and related patterns. We believe that this criterion must be examined and exposed in security pattern classifications since we often observed that several patterns may meet the same security property.

From these observations, we propose to contribute in the security pattern classification by proposing a classification methodology, composed of several successive steps, which lead to a pattern classification based upon security attacks, since the notion of attacks is usually well-known by designers and developers. More precisely, the contributions of this paper can be summarised by the following points:

- we propose a classification, which is based upon relationships among attacks, weaknesses, security principles and security patterns, expressing the pattern combinations that can be used to mitigate an attack. The classification is composed of security properties found in well established security bases i.e. the CAPEC (Mitre corporation, 2015a) and CWE (Mitre corporation, 2015b) bases. The classification is stored in a database available in (Regainia et al., 2016b). For readability purposes and to illustrate the navigability among patterns, the classification is illustrated by means of Attack-Defence Trees (ADTrees (Kordy et al., 2012)), which illustrate for a given attack its (more concrete) sub-attacks and the defenses, expressed here with security patterns combined with logic operations,
- our classification reveals the combinations of patterns that should be chosen to block an attack, and particularly it provides inter-pattern relations. We indeed take advantage of the studies about the inter-pattern relations (Yskout et al., 2006; Fernandez et al., 2008) and include them in the classification. For instance, these inter-pattern relations offer the advantage to make apparent the conflicting or alternative patterns,
- we propose a methodology, built on five steps, which infer relationships between attacks, weaknesses, security principles and patterns to generate the classification and Attack Defence Trees. The methodology takes as inputs CAPEC attacks, builds a hierarchical tree of attacks showing the sub-attacks of a given attack and links them with the targeted weaknesses found in the CWE base. Afterwards, we reuse an earlier work, proposed in (Regainia et al., 2016a), to build relationships among weaknesses, security principles and security patterns. All these steps provide the detailed justifications of the resulting classification. As most of the steps are automatic, this methodology can be followed again to update the classification.

We have limited our classification to the field of Web applications, which means that our pattern classification includes 215 CAPEC attacks, 136 CWE weaknesses, and 26 security patterns covering varied security aspects.

The remainder of the paper is organised as follows: Section 2 presents some related work and the motivations of the proposed approach. Section 3 recalls some security notions used in the paper. We introduce the methodology in Section 4 and illustrate its steps with the attack “CAPEC-39: Manipulating Opaque Client-based Data Tokens“. We give a short presentation of the classification and discuss about its advantages and limitations in Section 5. We traditionally conclude and give some perspectives for future work in Section 6.

2 RELATED WORK

Some taxonomies and classifications were proposed in the literature to help designers in the choice of the most appropriate security patterns with regard to a given context.

Bunke et al. presented a systematic literature review of the papers dealing with security patterns between 1997 and 2012. They finally proposed a classification based upon the application domains of patterns (software, network, user, etc.)(Bunke et al., 2012), but the notions of attacks or vulnerabilities are not mentioned. Vulnerabilities are taken into consideration for pattern classification in (Anand et al., 2014; Alvi and Zulkernine, 2011). This gives another point of view helping designers in the choice of patterns to fix software vulnerabilities. This vulnerability based classification seems meaningful since vulnerabilities are the natural causes of attacks on software systems, but the vulnerability concept is not the most known by designers.

This is why other authors proposed security pattern classifications, which refer to the attack concept

(Wiesauer and Sametinger, 2009; Tøndel et al., 2010; Alvi and Zulkernine, 2011; Uzunov and Fernandez, 2014). Wiesauer et al. initially presented in (Wiesauer and Sametinger, 2009) a short taxonomy of security design patterns made from links found in the textual descriptions of attacks and the purposes and intents of security patterns. Tondel et al. presented in (Tøndel et al., 2010) 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 methodology 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 (Alvi and Zulkernine, 2011). They analysed some security pattern templates available in the literature and proposed a new template composed of the essential elements needed for designers. They manually augmented the CAPEC attack documentation with a section named "*Relevant security patterns*" composed of some patterns (Alvi and Zulkernine, 2011). After inspecting the CAPEC base, we observed that this section is seldom available, which limits its use and interest. Finally, Uzunov et al. proposed in (Uzunov and Fernandez, 2014) a taxonomy 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 (Uzunov and Fernandez, 2014). They considered that the use of the CAPEC base is cumbersome and assumed that their threat patterns are abstract enough to encompass security problems related to the specific context of distributed systems (Uzunov and Fernandez, 2014).

Open Issues and Contributions

Alvi et al. outlined 24 of these classifications in (Alvi and Zulkernine, 2012) 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 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.

We also observed that the main issue of the above works lies in the lack of a precise methodology to build the classification. All of them are based upon the interpretation of different documents, which are converted to abstract relationships. The first consequence is that understanding the structure of categories and relationship is sometimes tricky. In addition, it becomes very difficult to extend these classifications.

Furthermore, we noticed that these classifications lack of navigability among patterns, which is an important property defined as the ability to guide the choice of designers among related patterns (Alvi and Zulkernine, 2012). We believe that a security pattern classification must be built from several security properties e.g., weaknesses, security principles, etc. to make the pattern choice more precise. The inter-pattern relationships should also be given: for instance, the conflicts among patterns, which may lead to inconsistencies in an application, must be noticeable.

In (Regainia et al., 2016a), we proposed a first semi-automatic methodology of classification and the classification itself, which exposes relationships among 185 software weaknesses of the CWE base (Mitre corporation, 2015b), security principles and 26 security patterns. It expresses which patterns partially mitigate a given weakness with respect to the security principles that have to be addressed to fix the weakness. This methodology is composed of some manual steps subdividing weaknesses and patterns into detailed security properties. Then, these are automatically associated together with respect to security principles. We organised the latter into a hierarchy to precisely express the security objectives of these properties.

In this paper, we continue the line of work previously initiated and present a methodology of classification to categorise the security patterns that can be used to mitigate attack patterns of the CAPEC base. Our classification aims at proposing a precise and accurate mapping between security patterns and CAPEC attacks: it is more accurate in the sense that we focus on the sub-properties of the security patterns and attacks and establish relations among these properties. In addition, the classification is completed with the inter-pattern relationships found in (Yskout et al., 2006). This is why we claim that the proposed classification is more precise.

But, the strong contribution of this paper lies in the presentation of the methodology itself. This one is

based on several automatic steps that can be followed again to complete the classification. We also propose the generation of Attack-Defense trees (ADTrees) putting CAPEC attacks as attack nodes and security patterns as defense nodes. These trees illustrate two points of view: they show the choice of an attacker (choice of attack, achieved with more concrete attacks) and the available mitigations that a designer can choose to devise its application. These are generated after the choice of an attack in the classification and are hence up-to-date w.r.t. the latter. Our ADTrees also express the relation among patterns (dependence, conflicts, etc.).

3 BACKGROUND

The proposed classification methodology is mainly based upon three security concepts: security patterns, CAPEC attacks and CWE weaknesses. We recall basics on these concepts below.

3.1 Security Patterns

A security pattern is a generic solution to a recurrent security problem, which is characterised by a set of structural and behavioural properties (Fernandez, 2007). It can be presented textually or with schema, e.g. UML diagrams. The quality of a pattern and its classification can be established with *strong points*, which correspond to sub-properties of the pattern. These properties characterize the forces and the consequences brought by the use of the pattern against a security problem (Harb et al., 2009). Strong points are manually extracted from the forces and consequences of a security pattern.

In addition, a security pattern can be documented to express its relationships with other patterns. Such annotations 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 (Yskout et al., 2006):

- "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.

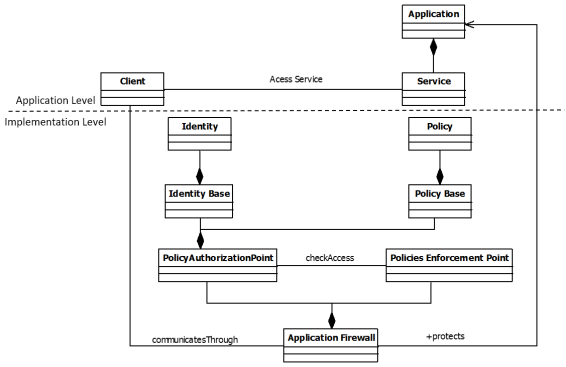


Figure 1: Application Firewall pattern.

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 pattern is related to two other security patterns (Yskout et al., 2006): it is an alternative to the patterns "Input Guard" and "Output Guard" since the application firewall is able to filter input calls, and also output responses from the application.

3.2 CWE Weaknesses

The CWE base (Mitre corporation, 2015b) 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.

3.3 Capec Attacks

The Common Attack Pattern Enumeration and Classification (CAPEC) is an open database offering a catalogue of attacks in a comprehensive schema (Mitre corporation, 2015a). *Attack patterns* are descriptions of common approaches that attackers take to attack software. An attack pattern, which we refer here as a document, consists of several sections; among them, a section describes the attack execution flow on vulnerable systems, other sections give the prerequisites,

the severity, the impact, the required attacker skills, etc.

In our context, three sections sound particularly interesting for starting a classification: the section "Related attack patterns" shows some relationships among attacks, the section "Related Weaknesses" lists the CWE weaknesses targeted by the attack and the section "Relation security principles" catalogues some principles defined as desirable properties targeted by the attacks.

The attacks of the section "Related attack patterns", are characterised by a type and a relation. The former expresses a level of abstraction of the attacks. Different levels can be found, giving a hierarchical structure: We list them from the more to less abstract as follows:

1. **Category:** at this level, an attack pattern expresses the attack mechanisms/domains from a high point of view (Injection, Target analysis, Social engineering, etc.),
2. **Meta Pattern:** as a refinement of Category, meta patterns express the possible specialisations of attack mechanisms without giving details neither the steps of the attack, nor the possible countermeasures,
3. **Standard Pattern:** gathers more detailed attacks, i.e. the attack steps, countermeasures and the related CWE weaknesses of the attacks,
4. **Detailed Attack Pattern:** this lowest level of abstraction gathers the specialisations of standard attacks in some specific contexts or needs. For example, the "*CAPEC-7 Blind SQL Injection*" is a special case and therefore a *child of* the standard attack "*CAPEC-66: SQL Injection*".

The section "Related attack patterns" gives binary relations between two attacks (a,b). These relations can be:

- "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 the attack "b",
- "a" can precede/can follow "b": when the attack "a" and "b" are put in sequence.

4 CLASSIFICATION METHODOLOGY

This methodology aims at inferring relationships among security attacks and security patterns, expressing which set of patterns can mitigate a given attack, completed with the relations among the patterns. Without loss of generality, we applied the following methodology on Web application attacks, but it can be applied on other kinds of applications.

After having studied the CAPEC base, we observed that attacks are described with a set of CWE weaknesses, a set of security principles and potential solutions and mitigations. These security activities can also be found in our previous classification (Regainia et al., 2016a), connecting weaknesses with security patterns. However, we noticed that the mitigations and security principles available in the attack documents often have a high level of abstraction making their use difficult. Furthermore, these are seldom supplied with attacks. As a consequence, to devise this classification with precision, we chose to decompose attacks into sub-properties, i.e. every attack is associated to its targeted weaknesses. We hence consider that a security pattern (or a set of patterns) is a solution to protect an application against a given attack if it is also the solution of a weakness targeted by the attack.

Our methodology is divided into five automatic steps, illustrated in Figure 2. In the first one, we collect the attacks of the CAPEC base and organise them into a hierarchy, from the more to the less abstract ones. In Step 2, we collect the relationships between every attack and CWE weaknesses, reflecting which weakness is targeted by an attack. In Step 3, we reuse our earlier classification (Regainia et al., 2016a) to extract for every CWE weakness, the security principles, mitigations and security patterns which fix the weakness. After the consolidation of the different databases built in the previous steps, we obtain a database DB_f from which the classification is automatically extracted in Step 4. Finally, for a given attack, we depict with an ADTree the attack associated with its sub-attacks and with defenses in the form of security patterns, which are themselves combined by means of logic operations.

Each step of our proposed methodology is detailed below and illustrated with the attack "CAPEC-39: Manipulating Opaque Client-based Data Tokens", which expresses a threat on applications using tokens, e.g. cookies, which hold client data.

4.1 Step 1: CAPEC Attack Extraction and Organisation

In this step, we want to extract the attacks found in the CAPEC base and organise them into a single tree, which describes a hierarchy of attacks from the most abstract to the most concrete ones so that, we can have, for a given attack, all its sub-attacks. To reach that goal, we rely on the relationships among attack

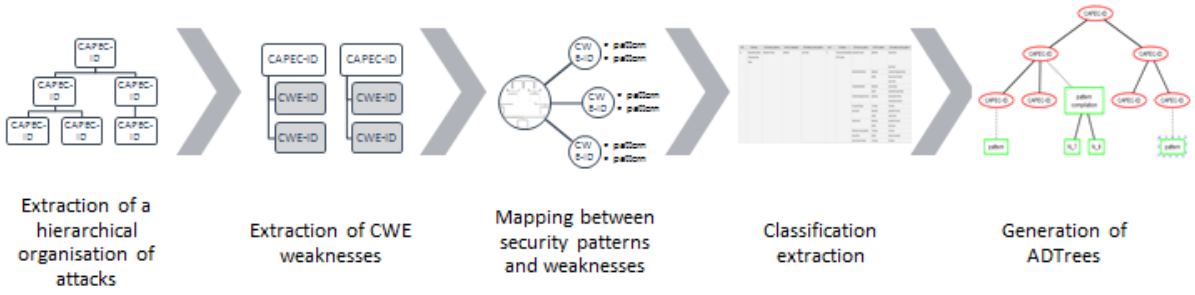


Figure 2: Classification methodology.

Related Attack Patterns

Nature	Type	ID	Name
ChildOf		22	Exploiting Trust in Client
ChildOf		223	Probabilistic Techniques
ParentOf		31	Accessing/Intercepting/Modifying HTTP Cookies

Figure 3: Hierarchical organisation of attacks for the attack CAPEC 39.

patterns found in the CAPEC section "*Related Attack Patterns*" (CAPEC base, Version 2.8). Figure 3 presents an example for the attack CAPEC-39. The abstraction level of the attack is expressed in the column "Type" (C stands for Category, S for Standard pattern), the links with other attacks by "Nature".

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".

We implemented this step with the tool Talend, [1] an ELT (Extraction, Load, Transform) tool which allows an automated processing of data independently from the type of its source or destination, by scanning the CAPEC attacks from the more abstract (those of the type "Category") to the most concrete ones (in the type "Detailed Attack Patterns") and we stored the resulting hierarchical tree into a database DB_1. This tree is currently composed of 215 attacks. The tree is composed of five levels w.r.t. the current CAPEC structure. If new attacks are added, our implementation can automatically take them into account to generate another tree.

4.2 Step 2: CWE Weakness Extraction from Attacks

We automatically extract for every CAPEC attack of the database DB_1 the CWE weaknesses targeted by the attack. These can be found in the section "*Related Weaknesses*" of the CAPEC documents. Weaknesses are grouped into two categories, "Targeted" and "Secondary" ranking the impact degree of the attack on a weakness. We only focused on the type "Targeted" even though it could also be relevant to consider both types.

The outcome of this systematic extraction is stored in a database DB_2, which encodes a mapping from the 215 attacks to 136 CWE weaknesses. Unsurprisingly, we observed that the attacks having a high level of abstraction (those of Category and Meta Pattern) are not related to any CWE weakness.

The attack CAPEC-39, taken as example, targets six CWE weaknesses, which illustrates here that the attack is indeed segmented into security sub-properties. Among them, we have "Reliance on Cookies without Validation and Integrity Checking" or "Improper Authorization", which reflect more precise sub-properties than the attack itself. These can be mitigated by several security patterns, which are revealed by means of the next step.

4.3 Step 3: Connection between CWE Weaknesses and Security Patterns

We proposed in (Regainia et al., 2016a) a classification, which exposes relationships among software weaknesses, security principles and security patterns. More precisely, for a given weakness, it provides its mitigations, the security principles that have to be followed to fix the weakness, the strong points (sub-properties) of security patterns that meet these principles and finally the security patterns allowing to correct the weakness. From this classification, we automatically extract, for every weakness found in DB_2 the following information:

- the complete hierarchy of security principles Sp related to a weakness, i.e. the arrangements of principles from the most abstract ones to the most concrete principles,
- for every principle sp in Sp, the set P_{sp} of security patterns associated with sp, the set $P2_{sp}$ of patterns not in P_{sp} that have relations with any pattern

[1] https://talend.com/

of P_{sp}, and the nature of these relations defined for couples of patterns by the annotations in {depend, benefit, impair, alternative, conflict}.

This automatic step produces the database DB_3.

4.4 Step 4: Data Consolidation and Classification Extraction

The databases obtained in the previous steps are integrated with the tool Talend into a single one, denoted DB_f, which is available on-line in (Regainia et al., 2016b). With DB_1, we have a hierarchical representation of attacks, which are eventually related to a set of CWE weaknesses given in DB_2. DB_3 encodes the relations among these weaknesses, the related security principles and security patterns. Hence, DB_f includes all the required information to expose several kinds of relations and classifications. For example, for a given attack, we can extract a hierarchical tree showing its sub-attacks. From this kind of extraction, attack trees (Schneier, 1999), could be generated. These show how a generic attack can be realised by more concrete attacks. Furthermore, the set of weaknesses targeted by an attack as well as the security principles that have to be followed to fix the weaknesses can also be selected from DB_f.

But, first and foremost, security patterns can be classified against attacks. We have chosen to catalogue the combinations of patterns that aim to mitigate a given attack, i.e. all the patterns that offer a mitigation for any weakness exploited by the attack. This step also automatically collects these combinations of patterns for every attack found in DB_f. More precisely, for a given attack, we extract:

- the information about the attack (name, identifier, description, etc.),
- the set of patterns P that are related to all the weaknesses targeted by the attack and the set of patterns $P2$ not in P that have relations with any pattern of P, and the nature of these relations.

Figure 4 depicts an extraction example for the attack CAPEC-39. The tabular gives the attack name, the security pattern allowing to block the attack (column 3), another alternative pattern (columns 4,5) and its sub-attack "CAPEC-31 Accessing/Intercepting/Modifying HTTP Cookies". The last columns give the security patterns allowing to overcome the attack CAPEC-31 and their relations with other patterns.

4.5 Step 5: Attack-Defense Tree Generation

We propose to greatly improve the readability of the classification, given in tabular form, by generating ADTrees, organising the attacks and the related security patterns. With ADTrees, attacks are illustrated with red nodes, which can be interconnected with the logic operations $and, / or$. An attack node can be mitigated with one defense node (in green squares) composed of sub defenses or one attack themselves combined the operations and / or.

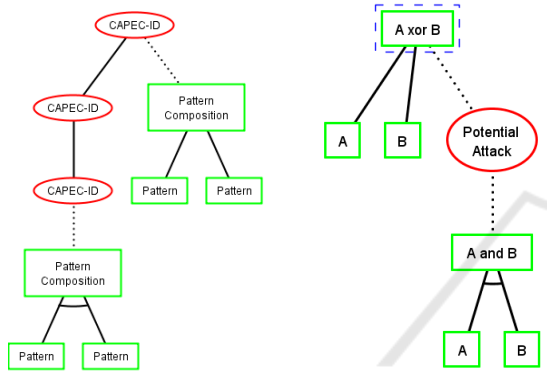
In our context, an ADTree shall be rooted by an attack chosen by a designer. This root node can be connected to other attack nodes, expressing sub-attacks, which can be connected to defense nodes, representing security patterns. Figure 5(a) illustrates a general example of ADTree. The "OR" operation is depicted with a group of classical edges, whereas "AND" is depicted with a group of classical edges connected with an arc.

ADTrees are generated by the following steps:

1. every CAPEC 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 if the attack has sub-attacks and so on until there is no more sub-attack. We obtain a sub-tree of the original hierarchical tree extracted in Step 1. All the attack nodes are here combined with the "OR" operation meaning that an attack can be accomplished if one of its sub-attacks is successfully done,
2. for every attack node A, we collect the set P of security patterns that mitigate the attack as well as the set $P2$ of security patterns having relations with some patterns of P. Given a couple of patterns $(p_1, p_2) \in P \cup P2$, we illustrate these relations with new nodes and logic operations. If we have:
   - $(p_1\ depend\ p_2)$ or $(p_1\ benefit\ p_2)$, we build three defense nodes, one parent node labelled by $p_1\ AND\ p_2$ and two nodes labelled by p_1, p_2 combined with this parent defense node by the AND operation,
   - $(p_1\ alternative\ p_2)$, we build three nodes, one parent node labelled by $p_1\ OR\ p_2$ and tw nodes labelled by p_1, p_2 combined with the parent defense node by the OR operation,
   - $(p_1\ impair\ p_2)$ or $(p_1\ conflict\ p_2)$, we want to use $(p_1\ XOR\ p_2)$ since the presence of p_2 decreases the efficiency or conflicts with p_1. Unfortunately, the XOR operation is not available with ADTrees. Therefore, we

LEV	LEV2 name	LEV2 Security_patterns	LEV2 SP_relationship	LEV2 related_Security_pattern	LEV3	LEV3 Name	LEV3 Security_pattern	LEV3 SP_relations	LEV3 related_Security_pattern
39	Manipulating Opaque Client-based Data Tokens	Application Firewall	alternative	Input Guard	31	Accessing/Intercepting/Mo HTTP Cookies	Application Firewall	alternative	Output Guard Input Guard
							Authentication Enforcer	alternative	Container Managed Security
								benifits	Secure Service Facade Secure Pipe
							Compartmentalization	alternative	Least privilege
								benifits	Distributed Responsibility
							Container Managed Security	alternative	Authorization Enforcer Authentication Enforcer
							Encrypted Storage	-No Value-	-No Value-
							Input Guard	alternative	Application Firewall
								benifits	Output Guard
							Output Guard	alternative	Application Firewall
								benifits	Input Guard
							Pathname Canonicalization	-No Value-	-No Value-
							Secure Pipe	benifits	Security Association
							Secure Service Facade	-No Value-	-No Value-

Figure 4: Data extraction for the attack CAPEC-39.



(a) Generic example of ADTree

(b) Conflicting pattern representation with ADTree

Figure 5.

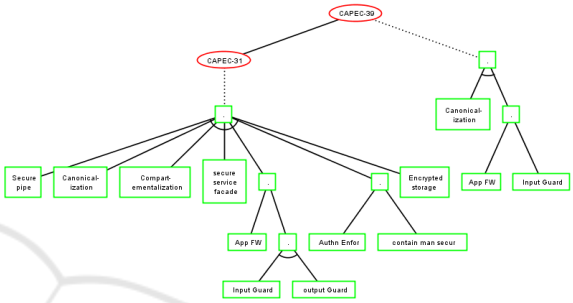


Figure 6: ADTree of the attack CAPEC-39.

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 generic sub-tree is depicted in Figure 5(b),

- p_1 having no relation with any pattern p_2 in $P \cup P_2$, we add the parent defense node labelled with p_1.

We may denote that "*Depend*" and "*Benifit*" relationships are presented two of them with the same "*and*"and this is explained by the fact that ADtrees do not allow to make contrast between these two types of relationships. The parent defense nodes, resulting from the above steps, are combined to a defense node labelled by "Pattern Composition" with AND. This last defense node is linked to the attack node A.

If we take back our example of attack, we obtain the ADTree of Figure 6, which shows that the attack CAPEC-39 has 1 sub-attack named "*CAPEC-31 Accessing/Intercepting/Modifying HTTP Cookies*". Because of lack of room, we presented in this example the security patterns directly related to the attack. An ADtree generated by our tool (Regainia, 2016) illustrates the patterns of the set P but also the set P_2 of patterns having a relation with those of P. Both of the two attacks target 17 weaknesses (6 for the CAPEC-39 and 11 for the CAPEC-31, which is more detailed). The attack and all its concrete forms can be mitigated by several combinations over 15 security patterns. For instance, the attack CAPEC-39 can be mitigated by two pattern combinations because the pattern "Application Firewall" can be replaced with "Input Guard". The number of security patterns related to both attacks CAPEC-39 and CAPEC-31 is explained here by the diversity of the targeted weaknesses. We assume for the classification generation that all of them have to be mitigated. As these ones cover different security issues here, e.g., input validation problems, privilege management, encryption problems, external control of the application state, etc., 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 mitigate this attack. In the worst case, an attack node

is not linked to a defense node, which means that either the classification is incomplete or the attack is relatively new and cannot be yet overcame by security patterns.

5 CLASSIFICATION DISCUSSION

Our current classification is an exemplary taxonomy built on a non exhaustive set of 215 CAPEC attacks, 26 security patterns and 136 CWE weaknesses related to Web applications. The classification and its complete list of elements is available in (Regainia et al., 2016b). Presented in tabular form, as illustrated in Figure 4, it enables multi-attribute based decisions insofar as patterns can be classified according to security principles, weaknesses and attacks.

The proposed classification complies with several quality criteria defined in (Alvi and Zulkernine, 2012). Among them, we have noted Navigability, Unambiguity and the Usefulness of the classification:

- Navigability, which is defined as the ability to direct designers among related patterns, is satisfied since we illustrate the classification with ADTrees. They indeed expose the hierarchical refinements of attacks and a combination of defenses that have to be applied to protect the application to design. The links between nodes exhibit the relationships considered in each step of the methodology,
- Unambiguity is taken into account since the classification is clearly defined by means of the methodology steps, which provide relations among attacks, weaknesses, security principles and security patterns. All these steps and those given in (Regainia et al., 2016a) justify the soundness of the classification. In addition, the classification provides the relationships among patterns, which help choose a correct combination of patterns, i.e. a conflicting combination can be avoided, the required patterns of another pattern are given,
- we believe the classification can be used in practice since it is based upon the CAPEC and CWE bases and of several security patterns presented in (Fernandez, 2007; Slavin and Niu, 2016). In addition, the ADTree formalism is one of the most prominent security formalism for analysing threats, it is supported by tools (Kordy et al., 2013) for editing, analysing and transforming them. Our ADTree generator actually generates XML files taken as inputs by these tools.

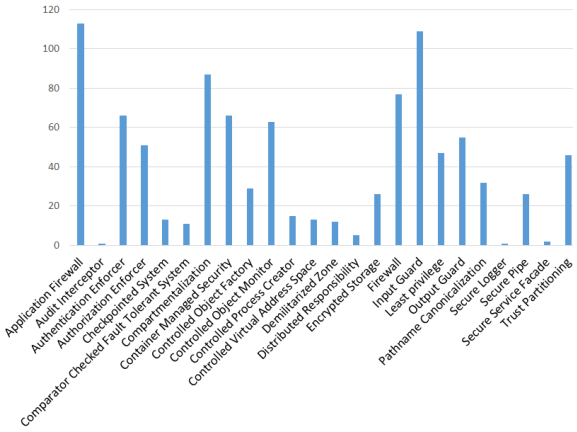


Figure 7: Number of fixed attacks per pattern.

Furthermore, a variety of statistical information can be automatically extracted from our classification, e.g., the ratio of weaknesses to attacks, of patterns to attacks, etc. For instance, Figure 7 also shows the number of attacks at least partially mitigated per pattern. Keeping in mind, that the set of patterns taken into consideration is not exhaustive, we can observe that 2 patterns seem to emerge for partly fixing a large part of the 210 attacks covered by the classification: “Input Guard“ and “Application firewall“, can overcome 113 and 109 attacks respectively. This kind of information shows that, from our classification, some useful outcomes can be extracted to 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 software design since they both partially block numerous attacks.

Our classification and methodology 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. In the first case, an attack combination can be represented in the CAPEC base as a sequence, which is given in a specific CAPEC section called *“Attack execution Flow“*. Our classification does not yet store and use the notion of ordered events. In the second case, if the attack has its own identification in the CAPEC base, it can be used with our methodology.

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 addition of new security patterns or weaknesses requires some manual steps. Our previous classification, proposed in (Regainia et al., 2016a), as-

sociates weaknesses and security patterns from documentation: if a new security pattern has to be added, two steps have to be manually done (mapping between the pattern and its strong points, and mapping between strong points and security principles). In the same way, if a new CWE weakness is added, two other steps must be manually completed (mitigation extraction, mapping between mitigations and security principles). The re-generation of the whole classification, which includes new attacks, weaknesses or security patterns is automatically performed. It could relevant to investigate whether some text mining techniques would help partially automate these manual steps without adding ambiguity.

6 CONCLUSION

In this paper, we have presented a classification methodology putting together CAPEC attacks, CWE weaknesses and security patterns to guide designers in their pattern choices. Given an attack, the classification provides a hierarchical tree of its sub-attacks (up to the most concrete ones), the targeted weaknesses, the security principles that have to be addressed to fix the weaknesses and the combinations of patterns that overcome the attacks. The classification is available in (Regainia et al., 2016b). ADTrees are automatically generated from the classification to ease its readability. For each attack of the classification, they portray its sub-attacks and combinations of security patterns. These ADTrees can be taken as a first step of other security processes, e.g., threat modelling.

Our most immediate line of future work is related to a specific section of the CAPEC base, called *"Attack execution Flow"*, listing the sequences of attacks (not the sets) that have to be followed to execute a meta-attack. We intend to take this section into consideration to extend the classification and the generation of extended ADTrees so that the latter explicitly show these attack sequences. The resulting trees (called SAND trees) shall increase the expressiveness of the ADTrees by adding the notion of ordered events. Then, from these SAND trees and the information included in the classification, we will focus on the generation of (generic) test cases to check whether an implementation is protected against the attacks or if security patterns are correctly contextualised and implemented w.r.t. the application context.

REFERENCES

Alvi, A. K. and Zulkernine, M. (2011). A Natural Classification Scheme for Software Security Patterns. *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 113–120.

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

Anand, P., Ryoo, J., and Kazman, R. (2014). Vulnerability-Based Security Pattern Categorization in Search of Missing Patterns. *2014 Ninth International Conference on Availability, Reliability and Security*, pages 476–483.

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

Fernandez, E. B. (2007). Security patterns and secure systems design.

Fernandez, E. B., Washizaki, H., Yoshioka, N., Kubo, A., and Fukazawa, Y. (2008). Classifying security patterns. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4976 LNCS, pages 342–347.

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

Kordy, B., Kordy, P., Mauw, S., and Schweitzer, P. (2013). *ADTool: Security Analysis with Attack–Defense Trees*, pages 173–176. Springer Berlin Heidelberg, Berlin, Heidelberg.

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

Mitre corporation (2015a). Common attack pattern enumeration and classification, url:https://capec.mitre.org/.

Mitre corporation (2015b). Common weakness enumeration, url:https://cwe.mitre.org/.

Regainia, L. (2016). Attack defence trees generator, url:http://regainia.com/adtreegen.zip.

Regainia, L., Salva, S., and Bouhours, C. (2016a). 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*.

Regainia, L., Salva, S., and Bouhours, C. (2016b). Security pattern classification url: http://regainia.com/research/database.html.

Rodriguez, E. (2003). *Security Design Patterns*, volume 49.

Schneier, B. (1999). *Attack trees: Modeling security threats*. Dr. Dobb's journal.

Schumacher, M. and Roedig, U. (2001). Security Engineering with Patterns. *Engineering*, 2754:1–208.

Slavin, R. and Niu, J. (2016). Security patterns repository, url: http://sefm.cs.utsa.edu/repository/.

Tøndel, I. A., Jensen, J., and Røstad, L. (2010). 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.

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

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

Yskout, K., Heyman, T., Scandariato, R., and Joosen, W. (2006). A system of security patterns.

Yskout, K., Scandariato, R., and Joosen, W. (2012). Does organizing security patterns focus architectural choices? *Proceedings - International Conference on Software Engineering*, pages 617–627.