Using Data Integration to Help Design More Secure Applications

Sébastien Salva(✉) and Loukmen Regainia

LIMOS CNRS UMR 6158, Clermont Auvergne University,
Clermont-Ferrand, France
{sebastien.salva,loukmen.regainia}@uca.fr

Abstract. Security patterns are reusable solutions, which enable the design of maintainable systems or applications that have to meet security requirements. The generic nature of security patterns and their growing number make their choices difficult, even for experts in software design. We propose to contribute in this issue by presenting a methodology of *security pattern classification* based upon *data integration*. The classification exhibits relationships among 215 software attacks, 66 security principles and 26 security patterns. It expresses pattern combinations, which are countermeasures to a given attack. This classification is semi-automatically inferred by means of a data-store integrating disparate publicly available security data. Besides pattern classification, we show that the data-store can be used to generate *Attack Defence Trees*. In our context, these illustrate, for a given attack, its sub-attacks, steps, techniques and the related defences given under the form of security pattern combinations. Such trees make the pattern classification more readable even for beginners in security patterns.

Keywords: Security patterns · Classification · Attack
Attack defence tree

1 Introduction

In the domain of software security, many documents (knowledge bases, papers, etc.) are now publicly available to help developers design and code more secure applications. For instance, the notion of security patterns, which is one of the topics of this paper, aims at providing guidelines to help in design secure systems [17]. Schumacher postulates that *a security pattern intuitively relates countermeasures to threats and attacks in a given context* [11]. As developers cannot be expert in all security fields, this plethora of (often complex) documents exposes them to the difficult choice of the most suitable security solutions for a given context. From these resources, several works recently proposed to organise them in order to help developers in their understanding and usage. Security patterns were arranged into different categories, e.g., by security principles [3,18], by application domains [4] (software, network, user, etc.), by vulnerabilities [2] or by attacks [2,15].

Despite the benefits brought by these classifications, they all are confronted to several limitations, which prevent their adoptions in the industry. Firstly, these classifications were manually devised, by directly comparing textual descriptions of different security concepts (patterns, principles, vulnerabilities, attacks, etc.). As these descriptions are generic and have miscellaneous abstraction levels, the categorisation of a pattern can be performed only when there is an evident relation between it and another security property. In addition, as these classifications are not *deterministic* (no strict definition of the classification process [3]), it often becomes 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* [19]. We indeed believe that many security pattern classifications lack of Navigability and Comprehensibility, which are quality criteria, proposed in [3] 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 expert developer.

From these observations, we propose to contribute to the security pattern classification by proposing a strict and precise classification process based on the concept of data integration. To make this classification navigable and comprehensible, we propose to automatically infer attack-defence trees (ADTrees [7]), which illustrate the security pattern combinations that can be used to prevent an attack on an application. More precisely, the contributions of this paper can be summarised by the following points:

– we propose a data integration methodology, built on six steps. These extract data from various Web and publicly accessible sources and store them into a data-store composed of relationships among attacks, attack steps, security principles and security patterns;
– we automatically derive a security pattern classification from the data-store, which can be updated after every data modification. For an attack, the classification expresses the security pattern combinations that can be used in the software design stage to later prevent the attack from being successfully carried out on the application;
– we generate Attack-Defence Trees (ADTrees [7]), which aim at supplementing the classification with illustrations depicting, for a given attack, its (more concrete) sub-attacks, steps and techniques along with defences preventing the attacks expressed here with security patterns combined with logic operations. Such ADTrees aim at improving the navigability and understanding of the previous classification.

We have generated a data-store and a security pattern classification specialised to the Web application domain, which is composed of 215 attacks, 26 security patterns and 66 security principles covering various security aspects. This classification and the ADTree generator are available here[1].

[1] http://regainia.com/research/database.html.

The remainder of the paper is organised as follows: Sect. 2 presents some related work and the motivations of our approach. The method, which aims at integrating data to build a data-store, is given in Sect. 3. Section 4 shows how we automatically extract the pattern classification and ADTrees from the data-store. We finally discuss on the resulting classification and conclude in Sects. 5 and 6.

2 Related Work

Several classifications were proposed to ease the pattern choice in the catalogues available in the literature, e.g., [1,19], totalling around 180 patterns. The classifications proposed in [2,12,13,15] focus on the attacker side. This choice of categorisation seems quite interesting and meaningful as attacks are more and more known and examined by designers. Wiesauer et al. initially presented a short taxonomy of security design patterns manually made from links between attack textual descriptions and security pattern purposes [15]. Tondel et al. presented in [12] the combination of three formalisms of security modelling (misuse cases, attack trees and security activity graphs) in order to give a more complete security modelling approach. In particular, they link some activities of attack trees with attacks; they also connect some activities of SAGs (security activity graphs) with security patterns. The relationships among security activities and security patterns are manually extracted from documentation and are not explained. Alvi et al. presented a classification scheme for security patterns putting together attacks and security patterns [2]. They analysed some security pattern templates available in the literature and proposed a new text section for completing the CAPEC classification [9]. After inspection, we observed that this section is seldom available, which limits its interest. Finally, Uzunov et al. proposed a taxonomy of security threats and patterns specialised for distributed systems [13]. This classification includes a library of threats and their relationships with security patterns.

Some papers reviewed these classifications and established a comparative study to point out their positive and negative aspects. Alvi et al. outlined 24 pattern classifications, including security pattern classifications, and established a comparative study to point out their positive and negative aspects [3]. 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 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. 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 [4]. They finally proposed a classification based upon the application domains of patterns (software, network, user, etc.).

We observed that the main problem of the above classifications lies in the fact that these all are manually conceived by directly finding relations in textual documents. Justifying these classifications or updating them is difficult. We also observed that they often lack of either Navigability or Comprehensibility or both. 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 [2], we propose a pattern classification expressing which patterns can be used to counter an attack step. 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. 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 inter-pattern relationships. Our data integration process also offers the advantage to justify the pattern classification and reduces the efforts required to update it. Finally, the generation of ADTrees makes the classification precise and readable even for novice in patterns or security.

3 Data Integration

We present below the architecture of the data-store we devised and an example of data integration for attacks and security patterns related to the Web application domain. Beforehand, we recall some basic fact about security patterns.

3.1 Security Patterns

Security patterns provide guidelines for secure system design and evaluation [17]. They often are presented textually or with schema (UML diagrams) and are characterised by a set of structural and behavioural properties.

Several security pattern catalogues are available on the Internet and literature, e.g., [1,19], themselves extracted from other papers. The quality of a pattern and its classification can be established by means of its *strong points*, which are properties expressing pattern key design features. Besides, a security pattern may have different relationships with other patterns. These new properties may noticeably help combine patterns and not to devise unsound composite patterns. Yskout et al. proposed a listing of pattern relations with the following annotations [18]: “depend”, “benefit”, “impair” (the functioning of the pattern can be obstructed by the implementation of a second one), “alternative”, “conflict”.

Application Firewall is a security pattern example whose primary objective is to filter out undesired messages given or produced by an application, by means of access control policies. Figure 1 depicts the UML class diagram of this security pattern. This schema shows that it forces to structure an application in such a way that the filtering logic is centralised and decoupled from the functional logic of the application. This also corresponds to a strong point of the pattern.

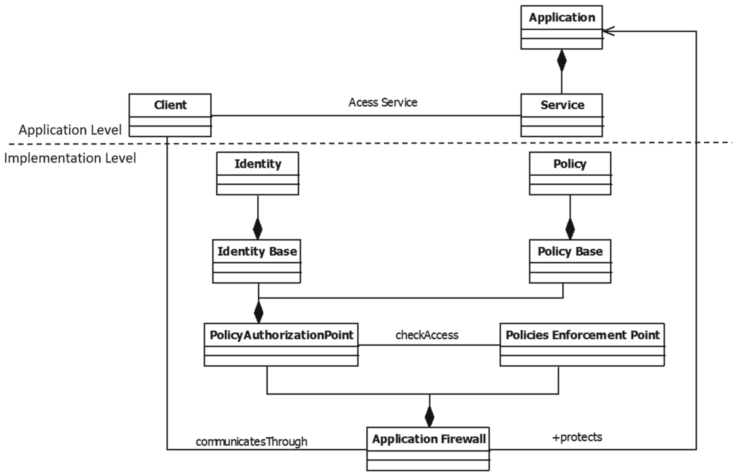


Fig. 1. Security pattern “Application Firewall”

3.2 Data-Store Architecture Presentation

The classification purpose is to ease the design of more secure applications. To do so, we propose to arrange security patterns in such a way that the resulting classification provides the set of patterns that can be used as countermeasures against a given attack (in reference to the security pattern definition of Schumacher [11]) and relations among patterns.

To infer a precise classification, we chose to anatomize attacks and security patterns into more detailed properties that can be interconnected in an explicit manner. After reviewing the literature and some attack bases, we observed that attacks are 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. We did not found smaller properties in the literature. On the other hand, security patterns can be characterised with some sub-properties, e.g., forces, consequences or strong points. A strong point is a pattern key feature that is extractable from its forces or consequences.

In both sides, countermeasures and strong points refer to the notion of attack prevention. But directly finding relations among them is still an obscure task as these properties have different purposes and abstraction levels. To solve this issue, we propose the option of gathering countermeasures into clusters (groups) to reach roughly the same abstraction level as strong points. Indeed, countermeasures are often much more detailed. Then, to link clusters and strong points, we chose to focus on security principles as mediators. We indeed observed that security patterns and strong points are classifiable w.r.t. security principles like most of the security techniques. Since countermeasures aim at preventing attack steps, it sounds natural that countermeasure clusters and strong points belong at least to one principle.

All the security properties considered here and their relations are structured with the meta-model illustrated in Fig. 2 as explained before. The entities refer to security properties, the relations formally express associations among them. This meta-model finally structures our data-store.

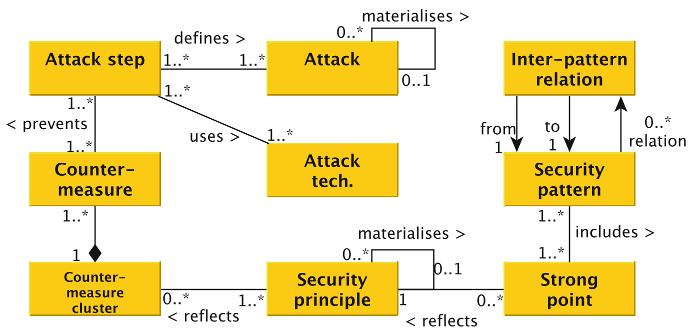


Fig. 2. The proposed mapping metamodel

3.3 Data Integration and Consolidation Steps

We present, in this section, the different steps for integrating security properties into the data-store. The data integration is divided into six steps, which aim at collecting security properties and establishing the different relations presented in Fig. 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 other ones as expected. These steps offer the strong advantage to semi-automatically achieve a data-store, which can be updated.

We implemented these steps mostly by means of Talend,[2] an ELT (Extraction, Load, Transform) tool that allows an automated processing of data independently from the type of its source or destination. We applied these steps on attacks, patterns and principles related to the Web application context. We provide some quantitative results related to this context with each step. But, these can also be applied to other kinds of systems as long as documentation is available. We integrate data coming from different sources: the CAPEC base [9], several papers dealing with security principles [8,10,14] and the pattern catalogue given in [19].

Step 1: Extraction of attacks, steps, techniques and countermeasures We chose to focus on the CAPEC base to extract information about security attacks. The Common Attack Pattern Enumeration and Classification (CAPEC) is an open database offering a catalogue of attacks in a comprehensive schema. Attack patterns are descriptions of common approaches that attackers 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 textual

[2] https://talend.com/.

sections. For instance, the section "Related attack patterns" shows interdependence among attacks, having different levels of abstractions.

We extracted attacks of the CAPEC base and organised them into a single tree, which 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". 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, which correspond to countermeasures.

This data extraction is automatically performed with a tool, which yields a database DB_1. From the CAPEC base Version 2.8, we 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, techniques, etc.

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. These countermeasures can be hence grouped into families to be later associated with a security principle.

We semi-automated this process by applying a hierarchical clustering technique of documents. We firstly used the tool *KHcoder*[3], which is a reputed tool performing quantitative content analysis or text mining. In short, we applied the tool as follows:

1. The Stanford (Part-of-speech) POS tagger 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 (the dissimilarity between sample sets) to measure a distance among countermeasures. The distance between two security controls is minimised when they have more common keywords.

Afterwards, we used the method Ward to automatically yield a hierarchy of countermeasure clusters [16]. We chose Ward because it offers the possibility to merge groups, piece by piece, instead of directly providing big clusters. In our case, this second solution would tend to build big clusters covering too much

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

security aspects, which would be later associated with too much security principles. Finally, the level to consider in the cluster organisation (and implicitly the number of clusters to keep) is manually chosen, as the choice of the number of clusters is always supervised with Ward. To get a coherent clustering, we chose the most suitable level after some iterations by checking whether the countermeasures obtained in the clusters refer to the same security principle or set of principles.

The resulting clusters are stored into the database DB_2. The 217 security controls collected by the previous step, are aggregated into 21 clusters.

Step 3: Security patterns and strong points integration
We manually collected security patterns and their strong points from the catalogue given in [19]. Strong points often have to be deduced in the sections referring to the forces and consequences of the patterns. 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 that can be shared with other patterns. For example, the patterns "Authorization enforcer" and "Container managed security" share the strong point "Providing the application with authorization mechanism";
2. The second relation is related to the annotations "depend", "benefit", "impair" or "alternative" defined among patterns [18]. With P a set of patterns, this relation is defined as a mapping from P^2 to the annotation set "*depend*", "*benefit*", "*impair*", "*alternative*", which provides for a pair of patterns $(p1, p2)$ an annotation about the relationship between $p1$ and $p2$.

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

Step 4: Security principle integration
We chose to organise security principles into a hierarchy, from the most abstract to the most concrete principles. We collected 66 security principles related to Web applications found in [8,10,14] and manually established dependencies in relation to the nature of each security principle, often described with text. The current hierarchy, which has four levels, is certainly not exhaustive. But it covers all the security patterns given in [19]. This security principle hierarchy is stored in the database DB_4.

This principle organisation gives a complete hierarchical view on security mechanisms, which are in the meantime required to prevent an attack step and which are provided by strong points. As principles are hierarchically organised from the most abstract to the most concrete ones, we can find relations between strong points and countermeasure clusters even if they do not exactly have the same level of abstraction.

Step 5: Mapping between strong points, security principles and countermeasure clusters
In this step, we established the many-to-many relation between strong points and security principles. This step was manually done because strong points and principles are mostly presented in an abstract manner. During this step, we observed that the abstraction level of the strong points better fit with the most concrete principles, which are the leaves of our hierarchical organisation.

In the same way, we established the many-to-many relation between countermeasure clusters and security principles. In Step 3, clusters include countermeasures sharing the same security aspects, e.g., Input validation, Authentication or Authorisation. Once these aspects are deduced, linking clusters and security principles becomes straightforward.

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

Step 6: Data consolidation
This automatic step integrates the previous databases DB_1 to DB_5 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.

This step is automatically performed by the tool Talend by means of the meta-model given in Fig. 2. The step produces the final database DB_f.

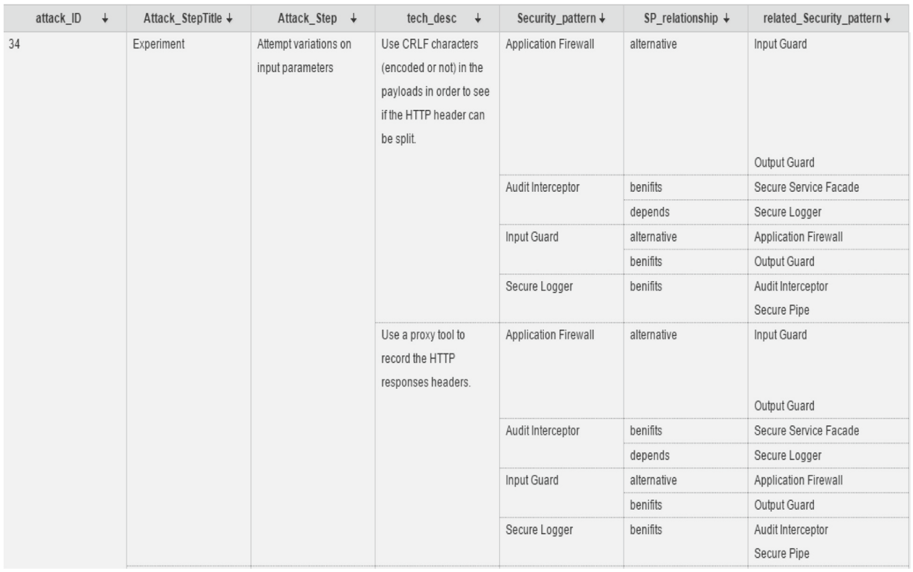
4 Security Pattern Classification and ADTree Generation

4.1 Security Pattern Classification

The database DB_f holds all the data and relations among attacks, steps, security principles and security patterns allowing to extract a security pattern classification. We have chosen to catalogue the combinations of patterns that are countermeasures against an attack. Given an attack *Att*, the following data and relations are hence extracted from DB_f:

- the information about *Att* (name, identifier, description);
- the tree $T(Att)$, whose root is *Att*, if *Att* is not a leaf of the attack 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. $Sp(st)$ represents the complete hierarchy of security principles related to a step, i.e., if a principle *sp* associated to the step *st* is not a leaf of our hierarchical organisation, then we also extract all the principle sub-tree whose root is *sp*;

– for each principle sp in $Sp(st)$, the set of security patterns P_{sp}, the set of patterns $P2_{sp}$ not in P_{sp} that have relations with any pattern of P_{sp}, and the nature of these relations defined for couples of patterns by the annotations in "*depend*", "*benefit*", "*impair*", "*alternative*", "*conflict*".



attack_ID	Attack_StepTitle	Attack_Step	tech_desc	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

Fig. 3. Data extraction for the attack CAPEC-34

Figure 3 depicts an extraction example for the CAPEC attack 34 "HTTP Response Splitting". The first column gives the ID of the chosen attack. This attack belongs to the category "Detailed" of the CAPEC, therefore it has no sub attacks (otherwise, the next columns would list them too). Columns 2 to 4 index the attack steps and techniques. Due to lack of room, 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 contextualised in the application model and implemented to prevent the attack. The last two columns add the security patterns being associated with the patterns of Column 5 and their relations. For instance, Fig. 3 shows that "Application Firewall" and "Input guard" are alternative patterns, hence using one of them is enough (although using both is not incorrect).

The classification extraction is achieved once all the attacks stored in the database DB_f are covered. This extraction is automatically performed with a tool based upon Talend. This tool can be re-executed every time the data-store is updated. The classification remains up-to-date accordingly.

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

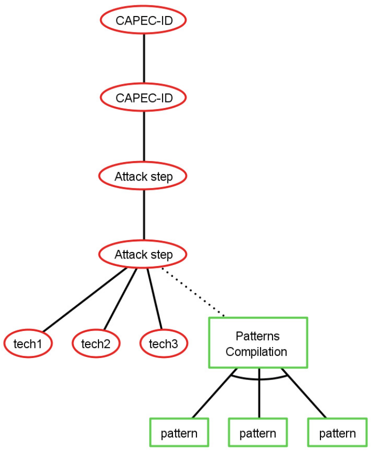
4.2 Attack-Defence Tree Generation

Attack Defence Trees "*are graphical representations of possible measures an attacker might take in order to attack a system and the defences that a defender can employ to protect the system*" [7]. We recall that ADTrees have two different kinds of nodes: attack nodes (red circles) and defence nodes (green squares). A node can be *refined* with child nodes using conjunctive or disjunctive refinements. The former is recognisable by edges going from a node to its children. The latter is graphically distinguishable by connecting these edges with an arc. Here, we extend these two refinements with the sequential conjunctive refinement of attack nodes, defined by the same authors in [5]. 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.

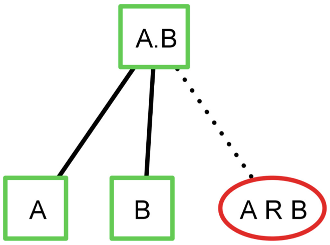
Keeping in mind that we use ADTrees to help developers design more secure applications, we propose to generate them with the general form illustrated in Fig. 4(a). This ADTree points out how an attack is sequenced with steps and how to prevent them with countermeasures given under the form of security pattern combinations. An ADTree root node is labelled by an attack. If the attack is linked to sub-attacks, the root node is also connected to child attack nodes expressing these sub-attacks. When an attack is defined with steps and techniques, its corresponding node has child nodes expressing them. A node labelled by an attack step has a child defence node, which is the root of a defence sub-tree expressing combinations of security patterns.

We automatically generate ADTrees from the data-store as follows:

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 with a disjunctive refinement if the attack has sub-attacks. This step is repeated for every sub-attack;
2. For each attack Att of 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 classification 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 are countermeasures of St. Given a couple of patterns $(p_1, p_2) \in P$, we illustrate these relations 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 defence nodes, one parent node labelled by $p_1\ R\ p_2$ and two nodes labelled by p_1, p_2 combined with this parent defence node by a conjunctive refinement;



(a) ADTree general form



(b) Conflicting pattern representation with ADTree

Fig. 4. General forms of the ADTrees generated by our approach. (Color figure online)

- (p_1 *alternative* p_2), we build three defence 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 $\{impair, conflict\}$. In this particular case, we would want to use the *xor* operation. Unfortunately, the latter is not available with the ADTree model. Therefore, we replace the operator by the classical formula $(A\ xor\ B) \longrightarrow ((A\ or\ B)\ and\ not\ (A\ and\ B))$. The *not* operation is here replaced by an attack node meaning that two conflicting security patterns used together might constitute a kind of attack. The node "Potential attack" expresses a kind of negation. The corresponding sub-tree is depicted in Fig. 4(b);
- p_1 having no relation with any pattern p_2 in P, we add one parent defence node labelled with p_1.

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

We implemented the ADTree generation with a tool, which takes as input an attack identifier and yields an ADTree, which is stored into an XML file. These files can be used with the editing tool given in [6]. As a consequence, ADTrees can be modified or updated as the developer wishes.

Figure 5 illustrates the ADTree obtained from the attack CAPEC 34. 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

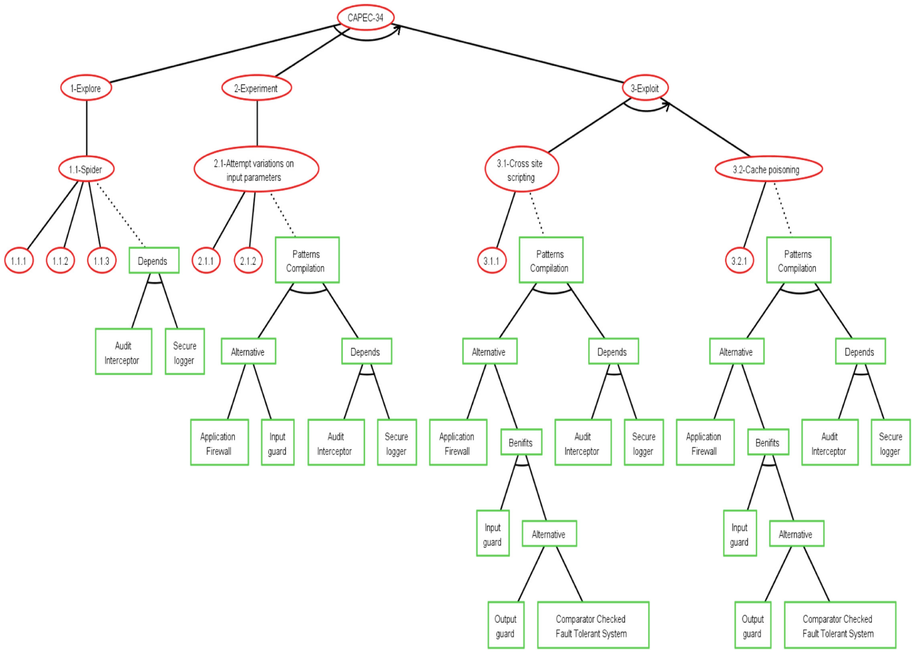


Fig. 5. ADTree of the Attack CAPEC-34

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. Defence nodes (square nodes) illustrate security pattern combinations. For instance, the step “1.1 Spider” refers to the Web application exploration through Graphical user interfaces in order to get all the URLs of the application. This step 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. This example illustrates that a designer can easily follow the concrete materialisations of an attack in an ADTree and can directly choose security patterns.

5 Classification Discussion

Our security pattern classification associates attacks, security principles and security patterns in order to help developers in the choice of the most suitable pattern combinations to design and code secure applications. The current classification is founded on 215 CAPEC attacks, 26 security patterns and 66 principles related to the Web application context. It enables multi-attribute based decisions insofar as patterns can be selected according to the provided inter-pattern relations and/or according to the attack steps.

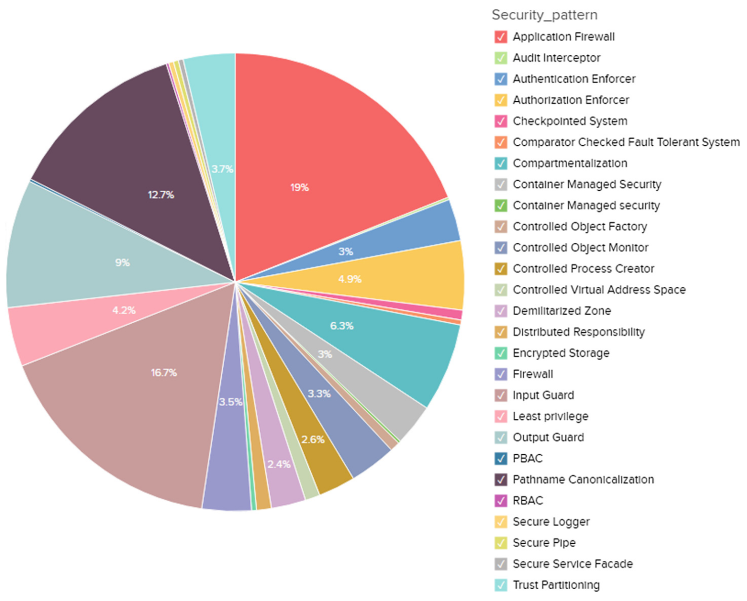


Fig. 6. Percentage of fixed attacks per pattern

Alvi et al. proposed in [3] some criteria for measuring classification quality. Among these criteria, we have noted that our classification meets:

- Navigability: our classification, accompanied by ADTrees, satisfies this criterion as it exhibits the hierarchical refinements of an attack and, for every attack step, the combinations of patterns, which should be integrated in the application model. In addition, the classification provides the relationships among security patterns, which help choose the most appropriate pattern combination;
- Determinism: the classification is clearly defined by means of the method steps. All these steps justify the soundness of the classification;
- Unambiguity/Comprehensibility: patterns are classified w.r.t. defined categories, i.e., attacks, steps, and security principles. 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 [19] and upon the CAPEC base, which is more and more employed in the industry;
- Repeatability: the classification is generic and can be reused. Furthermore, the data-store and the classification can be updated and generated semi-automatically.

Besides, a variety of statistical information can be automatically extracted from the data-store. For instance, Fig. 6 depicts a pie chart, which shows the ratios of attacks that can be partially prevented per security pattern. These kinds of charts, which are automatically generated from the data-store, seem quite useful to guide designers towards security analysis, good practices and education. For instance, with the above chart, a designer can observe that 2 patterns seem to emerge for partly countering a large part of the 215 attacks covered by the classification, namely "Input Guard" and "Application firewall". It is manifest that if we complete the data-store with more data, e.g., patterns or attack risks, such charts could be more refined and adapted to the developer needs.

Our classification and data integration process present some limitations, which could lead to future works. Firstly, we did not consider the notion of attack combination. Such a combination could be seen as several attacks or as one particular attack. Furthermore, the classification is not yet exhaustive: it includes 215 attacks out of 569 (for any kind of application) and 26 security patterns out of around 180. We also do not take into consideration the ADTree size. This is a strong limitation since large trees are usually unreadable, which contradicts the classification purposes. The ADTree reduction could be a first solution on this problem. But, reducing such trees remains a hard problem as the node meaning must be taken into account in the node aggregating process.

6 Conclusion

We have proposed a security pattern classification method associating attacks, security principles and security patterns in order to help designers understand the inner workings of attacks and choose the most suitable pattern combinations to design secure applications. This method integrates data obtained from various sources and subdivides attacks and patterns into detailed properties, which are associated in accordance with security principles. The pattern classification is then automatically generated from the data-store. The data-store and the classification can be upgraded by following some steps only. We also proposed to portray this classification by means of ADTrees showing attack scenarios (steps, techniques, etc.) and countermeasures given as security pattern combinations.

In future research, we will firstly focus on the automation of some data integration steps. Indeed, it could be relevant to investigate whether some text mining techniques would help partially automate the extraction and integration of the security pattern properties without adding ambiguity. As our ADTrees exhibit concrete attack scenarios composed of sequences of steps, we also intend to use them for the test case generation to check whether an implementation is protected against the attacks given in an ADTree or if security patterns are correctly contextualised and implemented w.r.t. the application context.

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

References

1. Security pattern catalog. http://www.munawarhafiz.com/securitypatterncatalog/
2. Alvi, A.K., Zulkernine, M.: A natural classification scheme for software security patterns. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 113–120 (2011)
3. Alvi, A.K., Zulkernine, M.: A comparative study of software security pattern classifications. In: 2012 Seventh International Conference on Availability, Reliability and Security, pp. 582–589 (2012)
4. Bunke, M., Koschke, R., Sohr, K.: Organizing security patterns related to security and pattern recognition requirements. International Journal on Advances in Security 5 (2012)
5. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 339–353. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_23
6. Kordy, B., Kordy, P., Mauw, S., Schweitzer, P.: ADTool: security analysis with attack–defense trees. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 173–176. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_15
7. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack-defense trees. Journal of Logic and Computation p. exs029 (2012)
8. Meier, J.: Web application security engineering. IEEE Secur. Priv. **4**(4), 16–24 (2006)
9. Mitre corporation: Common attack pattern enumeration and classification (2015). https://capec.mitre.org/
10. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proc. IEEE **63**(9), 1278–1308 (1975)
11. Schumacher, M.: Security Engineering with Patterns: Origins, Theoretical Models, and New Applications. Springer-Verlag New York Inc., Secaucus (2003)
12. Tøndel, I.A., Jensen, J., Røstad, L.: Combining misuse cases with attack trees and security activity models. In: International Conference on Availability, Reliability, and Security, 2010, ARES 2010, pp. 438–445. IEEE (2010)
13. Uzunov, A.V., Fernandez, E.B.: An extensible pattern-based library and taxonomy of security threats for distributed systems. Comput. Stand. Interfaces **36**(4), 734–747 (2014)
14. Viega, J., McGraw, G.: Building Secure Software: How to Avoid Security Problems the Right Way. Portable Documents, Pearson Education (2001)
15. Wiesauer, A., Sametinger, J.: A security design pattern taxonomy based on attack patterns. In: International Joint Conference on e-Business and Telecommunications, pp. 387–394 (2009)
16. Willett, P.: Recent trends in hierarchic document clustering: a critical review. Inf. Process. Manag. **24**(5), 577–597 (1988)
17. Yoder, J., Yoder, J., Barcalow, J., Barcalow, J.: Architectural patterns for enabling application security. In: Proceedings of PLoP 1997, vol. 51, p. 31 (1998)
18. Yskout, K., Heyman, T., Scandariato, R., Joosen, W.: A system of security patterns (2006)
19. Yskout, K., Scandariato, R., Joosen, W.: Do security patterns really help designers? In: Proceedings of the 37th International Conference on Software Engineering - Volume 1, pp. 292–302. ICSE 2015. IEEE Press, Piscataway (2015). http://dl.acm.org/citation.cfm?id=2818754.2818792