Managing Security Knowledge through Case based Reasoning

Corrado Aaron Visaggio and Francesca De Rosa

Dept. of Engineering, University of Sannio, Italy

Abstract. Making secure a software system is a very critical purpose, especially because it is very hard to consolidate an exhaustive body of knowledge about security risks and related countermeasures. To define a technological infrastructure for exploiting this knowledge poses many challenges. This paper introduces a system to capture, share and reuse software security knowledge within a Software Organization. The system collects knowledge in the form of misuse cases and makes use of Case Based Reasoning for implementing knowledge management processes. A reasoned analysis of the system was performed throughout a case study, in order to identify weaknesses and opportunities of improvement.

1 Introduction

Knowledge about software security is now acquiring an economic and strategic value for Organizations: recently a market of vulnerabilities is developing and expanding fast [4]. In order to improve security into software products, developing or hiring skilled professionals is not enough [11]. As pointed out by Barnum and McGraw [13], critical software security knowledge should be captured and widely shared. Once formalized and catalogued, this knowledge could be used within the Organization with two purposes: training, and supporting the problem solving process. Previous experience could be reused as is, or could help produce the solution for a new problem. Threats modeling is a central aspect of the security engineering process [5]. A way to model threats in terms of interaction with the system is the misuse case [9]. A misuse case describes potential system behaviors that are not acceptable by a system's stakeholders. A misuse case defines a sequence of steps which lead the user to misuse the system, i.e. to violate privacy or security policies. These misuses either represent high-probability attacks or high-impact events that negatively affect the system's legitimate stakeholders. Misuse cases should be at a level of detail that drives design activities, and they are convenient means for capturing knowledge about system's security. A misuse case could leverage a security flaw at three different levels of detail:

- domain level, i.e. when the user process allows illegal access to sensitive resources; for instance, when web pages that should be accessed with https protocol could be reached with a http connection, too;
- design level, i.e. when the design exposes security bugs; an example is the sql injection vulnerability;

Visaggio C. and De Rosa F. (2009).
Managing Security Knowledge through Case based Reasoning.
In *Proceedings of the 7th International Workshop on Security in Information Systems*, pages 127-135
DOI: 10.5220/0002200401270135

– technological level, i.e. when the bug is due to the specific technology (programming language, dbms, frameworks, api's, and so forth). An example of this kind of vulnerabilities is discussed in [2].

Of course, the misuse case could also exploit flaws concerning more than one level. With this paper we present a system for capturing, sharing, and reusing security knowledge into an Organization. The knowledge is formalized in the form of a misuse case and stored into a knowledge base. The system finds vulnerabilities which were successfully solved (and whose solution could be retrieved in the knowledge base) similar to a new one. If this similarity is enough high, the solution or parts of it could be re-applied to solve the current security problem. This usually happens when two vulnerabilities share one of the three levels but concern more than one level. For instance, the sql injection mechanisms do not depend from the technology, so a designer could re-use the same countermeasures, properly adapted, as well as when using asp, jsp or php (technology level) and when implementing different processes, i.e. different web applications' features (domain level). The paper is organized as follows: next section introduces the system; the third section discusses an example. Finally, conclusions are drawn.

2 The System

Our system relies on case base reasoning (CBR) [3] as model for knowledge storage and retrieval. A case is a couple (problem, solution). The case based reasoning is a problem solving technique which exploits the learning from similar cases in order to solve a new problem. The CBR process for problem solving is a four-steps cycle (Fig. 1).

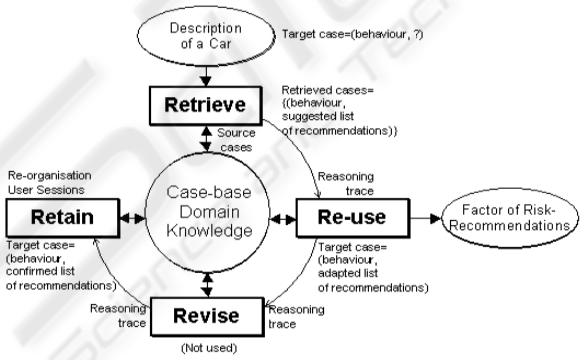

Fig. 1. CBR cycle.

Once the new problem is described (*new case*), the engine searches for *similar* problems stored into the base (***retrieve phase***) by calculating the similarity of the new case with the *previous cases*. Two cases are similar when they correspond to similar problems. Similarity functions are divided in two classes: the *surface similarity*, that expresses the distance between two cases by a number into a range [0,1] or [0,100]; and the *structural similarity*, that considers cases as complex structures, as well as graphs:

similarity is a function which compares the properties of these structures into the two cases. The retrieved case which has the highest value of similarity is the candidate for solving the new problem (***reuse phase***). Three classes of reuse exist: (i) replacing parts of the solutions, namely *substitution*; (ii) altering part of the structure, namely *transformation*; and finally (iii) applying the derivation of the (old problem's) solution to the new problem, namely *generative adaptation*. The proposed solution to the new problem, i.e. the *new case* is then validated (***revise phase***). Finally the new case must be integrated in the case base (***retain phase***).

With this paper we adapt the CBR mechanisms for capturing, sharing, and reusing knowledge about security threats within a Software Organization. We focused on the models applied to the retrieve phase. The structure of the case refers to the specifications of misuse case provided by Sindre et al. [7], which are detailed in the table 1, while a complete case is provided in table 8 (missing attributes are empty in the case).

The user will define the case in natural language, but special *literal value*s must be used when the case is filled in. Such values are named ***domain's tag***.

Table 1. Structure of a misuse case.

Name	Name of the Misuse Case
Summary	Brief description of the Misuse Case
Date	Generation Date of the Misuse Case
Author	Author of the Misuse Case
Basic Path	Main sequence of steps needed to accomplish the attack
Alternative Path	Alternative actions' sequence for the attack
Mitigation Points	Countermeasures for reducing the risks of the attack
Triggers	Events which could activate the misuse case
Preconditions	Characteristics and properties of the system necessary to make the attack possible
Assumptions	Conditions enabling the attack and which are external to the system
Mitigation Guarantee	Conditions to validate the mitigation of the threat
Related Business Rules	Business rules which are affected by the security flaws.
Stakeholders and Threats	Stakeholders and threats concerned by the misuse case
Potential Misuser Profile	Competence, skill, and capability needed for accomplishing the attack
Scope	Impact of the misuse
Abstraction Level	Design Portion interested by the misuse case
Precision Level	Architectural component interested by the misuse case

These values will be the elements of the corresponding attribute's domain. As a matter of fact, each attribute is defined upon a finite and discrete domain, which should increase over time. This happens because when the number of cases in the knowledge base gets bigger, the need of a greater expressiveness to describe misuse cases arises. For instance, in the example in table 8 the tags for the attribute *preconditions* are *publicly available* and *registered as a customer*.

Let O_target be the searched object in the case base; it describes the problem that the user needs to solve. O_target is a partially filled in case. As some attributes do not help the retrieve phase, the candidate attributes to be compiled in the O_target are: triggers, preconditions, assumptions, related business rules, stakeholder and threats, potential misuser profile, scope, abstraction level, and precision level. O_target is a matrix where each row represents an attribute of the misuse cases, and each column is a value assumed

by the attribute, i.e. a key for the search of similar cases. An exemplar O_target is shown in table 2.

The system will search the *most similar* cases in the knowledge base. In order to establish whether two cases are similar, a *similarity measure* must be defined. The similarity between two objects is a function, called Global Similarity and defined in the interval [0:1], where 1 corresponds to the maximum similarity. A similarity measure fulfills these properties: reflexivity, symmetry, monotony, and triangle equality. Let Oa_1, a_2, , a_n be a complex object with *n* attributes a_i, while let (O_1,O_2) be two instances of the O_α. First, the similarity between the correspondent couple of values for each attribute a_i of (O_1,O_2) should be calculated, namely *local similarity (localSim$_i$)*. Thus, the *global similarity* is calculated by including the local similarities for all the n attributes of the object. $GlobalSim\, O_\alpha\, (O_1, O_2) = \frac{1}{n} \sum_{i=1}^{n} LocalSim_i$. The way of local similarity calculation depends on the kind of objects' attribute. In case of: numbers, similarity is a distance; strings, similarity is an evaluated comparison; symbols, similarity is calculated for each possible combination; object, similarity is measured by a proper function which considers all the object's fields. The *soundness* of a similarity measure is expressed through the *gold standard*. This is a set of comparisons with a desired similarity value, defined by the user or a domain expert. A key point of estimating the quality of a similarity measure will always be the calculation of its deviation to the gold standard. This basically consists of two steps: choosing pairs of objects to compare and choosing a meaningful measure for calculating the *deviation*. Some algorithms have been proposed in order to accomplish the first step; as this is not the focus of this paper, this argument will be not discussed here. We used the formula: $\frac{1}{n} \sum_{i=1}^{n} |GoldStd_i - simValue_i|$, where n is the number of comparisons, goldStd$_i$ is the gold standard value and simValuei is the calculated value for the i-th comparison. Further methods includes the root mean square error and the threshold error, which will not be treated here. Finally the *fitness function* [16], which is a hyperbolic function must be defined as

$Fitness(deviation) = \frac{z}{deviationMax + a} - b$,
where:
$a = \frac{fitnessMean * deviationMax}{fitnessMax - 2fitnessMean}$
$b = \frac{fitnessMax * fitnessMean}{fitnessMax - 2fitnessMean}$
$z = a * (fitnessMax + b)$

- deviation$_{max}$, as the max measure of diversity, and varies between 1 and 100;
- fitness$_{mean}$, measures the quality of the comparison;
- fitness$_{max}$, measures the maximum of similarity.

These can be used to adapt the hyperbola to the concrete needs one might have, transforming a deviation to a fitness. These might be that a defined maximal deviation leads to a fitness value of 0 and that a deviation of 0 leads to a defined maximal fitness value(or infinity if a is chosen to be 0).

So if one defines two points which the hyperbola has to cross, namely, fitness(0) = fitness$_{max}$ and fitness(deviation$_{max}$) = 0, it is possible to set up two equations for the parameters (a, b and z) of the common hyperbola. So a third point of the hyperbola

is needed (i.e., can be chosen) to set up the third equation. Having these three points it is possible to calculate the three parameter values. Let's define a value fitness$_{mean}$, that corresponds to the fitness function's value for the deviation of deviation$_{max}$/2. Choosing this value to be fitness$_{max}$/2, the resulting function would be a straight line.

The similarity function consists of a collection of similarity tables, one for each attribute of the case. The similarity table defines the similarity between all the possible couples of that attribute's values. Let α= a_1, a_2, a_3, a_4,... a_k be an attribute and let α_i, with $i \in [1, k]$ be a possible value assumed by α. A similarity table, i.e. T_a, for the attribute α is a triangular table where each element on the l-th row and j-th column is the local similarity between the tags a_l and a_j, i.e. T_$a_{lj} = localSim_a(a_l, a_j)$

This is needed as the similarity between two values can be assigned only with regard to the semantics of the attribute. table 3 shows an exemplar excerpt of the similarity table for the Stakeholders and Threats attribute in the function f_2 (used in the next section's example). Local similarity values for the different tags are provided.

In summary, the CBR Retrieve phase process is recalled: first, the user defines the target object to search, i.e., by instantiating the matrix O_{target}.The system calculates the global similarity for each candidate case (O_{retr_j}) in the knowledge base, namely and GlobalSim (O_{target}, O_{retr_j}) . The system selects the O_{retr_j} which is able to maximize the fitness function.

The user can exploit a retrieved case $O_{retrieved}$ in order to solve the new problem. There are three situations. $O_{retrieved}$ fits well the new problem: the solution is applied to the problem (which is actually not a *new* one), i.e. knowledge is reused (***reuse phase***). $O_{retrieved}$ partially fits the new problem: the solution proposed by $O_{retrieved}$ can not be applied as is, but it could help user define the solution for the new problem: a new case is created and stored, i.e. the knowledge base is enlarged. Finally $O_{retrieved}$ is so different from O_{target} that it does not provide any help. In this latter situation, the existing knowledge is not enough to face the new problem. ***Revise phase*** consists of verifying that the solution is effective. Finally, the case is catalogued in the case base (***Retain phase***). If new attribute values are introduced with the new case, the similarity table must be properly updated. The next section will discuss an example of the retrieve and reuse phase.

3 An Example

Let's consider the following problem: how to mitigate the risk that passwords used to authenticate for restricted services are captured by other users or lost. The problem is formalised in table 2.

Table 2. An exemplar problem.

Trigger	Always true
Assumption	Passwords are used to authenticate
Related Business Rule	Restricted services
Stakeholders and Threats	Give away the password to other Potentially losing money

For comprehension' sake, let's assume that there are four candidate cases into the case base, namely the misuse cases #524, #530, #557, and #541. In order to understand how the system works, let's consider two different similarity functions, f_1 and f_2 . The example will show how similarity functions could affect the retrieval results. Each similarity function consists of a similarity table for each attribute used to define the problem. For space's reasons, only parts of the two functions are showed, in table 7. Some values in the similarity function f_1 are intentionally set wrong, in order to emphasize the effects in the retrieve phase. For instance, in the Related Business Rule attribute of f_2, similarity between the tag *Available over the internet* with itself corresponds to 0.1, while it should reasonably be 1.0.

Table 3. Similarity table for Stakeholders and Threats attribute belonging to the similarity function f_2.

	Loss of data	Potentially losing money	Give away the password to others	Alteration of data	Meeting with No-relevant people
Loss of data	1.0	0.3	0.2	0.7	0.1
Potentially losing money	0.3	1.0	0.3	0.2	0.1
Give away the password to others	0.2	0.3	1.0	0.1	0.2
Alteration of data	0.7	0.2	0.1	1.0	0.1
Meeting with No-relevant people	0.1	0.1	0.2	0.1	1.0

Table 4. Comparing retrieval results by applying the two similarity functions f_1 and f_2.

Misuse case ID	**f_1**		**f_2**	
	Fit. mean 0.05	**Fit. mean 1.00**	**Fit. mean 0.05**	**Fit. mean 1.00**
524	100	100	100	100
530	46	57	12	26
557	37	37	44	44
541	22	35	13	26

The *fitness mean* is a parameter for evaluating the quality of comparison. The higher this parameter is the better is the evaluation of the retrieved case. As a matter of fact, for both the functions, the values obtained by setting the parameter at 1.00 are higher than when the parameter is 0.05. Let's analyze now the results of the retrieve phase. In both the cases the misuse case #524 (see table 4) scored the maximum, which is 100. This case is perfectly correspondent to the problem description, i.e. the case will be reused as is, indeed. The #524 summary quotes: A crook obtains passwords for user accounts belonging to someone else, for the e-shop application typically e-shop clerks or system administrators. In order to get the complete picture of the differences, let's compare the misuse case #530 which is considered the worst one for f_2, with #541, that is the worst one for f_1 (see table 6).

The #541 regards disclosing the agreement about the date of the meeting to other people who are not authorized. The #530 describes the case when the misuser gains access to the system by trying large sets of passwords. Accordingly to f_1's results, #530 is much more suitable than #541. This evaluation is not satisfactory, as #530 description misses two attributes' value, i.e. the problem is much more general than the problem we need to solve, and consequently the solution, too. In conclusion the results provided by

Table 5. Similarity table for Stakeholders and Threats attribute belonging to the similarity function f_2.

Attributes	**Problem**	**Retrieved Case: #524**
Trigger	Always true	No Value
Assumption	Passwords are used to authenticate	Passwords are used to authenticate e-shop clerks and administrators
Related Business Rule	Restricted services	Only authorized users shall be able to access restricted services
Stakeholders and Threats	Give away the password to other	[]the crook may also sell or give away the password to others who have an interest in harming the e-shop [..]

f_2 are more realistic, as both #530 and #541 have a close similarity, while the similarity with the O_{target} is definitely low. Let's analyse briefly the points of strength and weakness of the solution presented here. Pros are: it is possible to manage security knowledge without introducing further structures, or tools. As a matter of fact, the system exploits directly misuse cases, that should be integrated in the security engineering process. The main drawbacks are related to the similarity functions. Maintenance is costly, as every change to the similarity tables affects other tables. Furthermore, if the similarity tables are not properly set up, the retrieval could be scarcely effective.

4 Related Work

At the best knowledge of the authors the problem of capturing and reusing security knowledge modeled as misuse case has been not faced, with the only exception of [17]. Ingalsbe et al. [8] introduce a process of threat modeling basically aimed at risk mitigation. Modeling the threats is used as a basis for evaluating related risks. This paper copes with the organizational aspects of threat modeling. Some authors [1] highlight the need for interleaving and aligning security engineering and software engineering processes. The paper does not face the problem of collecting knowledge about security risk mitigation. Authors in [14] present a unified threat model for assessing threats in web applications, by extending the threat tree model. They utilize historical statistical information contained in this model to design threat mitigation schemes. The threat assessing results and mitigation schemes should help direct secure coding and testing. In order to solve the problems of evaluating system security threat in the complex system, Liu and Liu [15] introduce a threat model based on the attacking-tree graph. First, an evaluating standard of the feasibility and harmful level of the vulnerability exploitation is given. Then an attacking-tree graph of the target system is constructed based on the relationship among exploitations of vulnerabilities. This model is able to calculate the impact of all kind of threats on the system security. Paper [12] presents an approach for addressing the threat modeling in pervasive computing; the model could also support the risk analysis. To improve trustworthiness of software design, paper [6] presents a formal threat-driven approach, which explores explicit behaviors of security threats as the mediator between security goals and applications of security features. To specify the intended functions, security threats, and threat mitigations of a security design as a whole, authors' method relies on aspect-oriented Petri nets as a unified formalism. All these papers focus on the problem of threat modeling. Paper [10] proposes a threat model-driven security testing approach for detecting undesirable threat behavior at run-

time. The threat model guides the code instrumentation; instrumented code is tested while the execution traces are collected and analyzed to verify whether the undesirable threat traces are matched. This paper applies threat modeling for strengthening security testing.

Table 6. Comparing #530 and #541 misuse cases.

	Problem	**Retrieved Case: #530**	**Retrieved Case: #541**
Trigger	Always true	Always true	Always true
Assumptions	Passwords are used to authenticate	No Value	Agreement is not encrypted
Related Business Rules	Restricted services	No Value	Information about the meeting should be available only to the concerned meeting participants
Stakeholders and threats	Give away the password to other possible	Possible loss of data; disclosure of data, possible alteration of data. May disrupt business and affect customer relations	No Value

Table 7. Comparing similarity tables of f_1 and f_2.

Similarity Function f_1

Related Business Rule	Available over the internet	Restricted services	Restricted access
Available over the internet	0.1	1.0	1.0
Restricted services	1.0	0.1	0.2
Restricted access	1.0	0.2	0.1

Assumption	Uses the network to log	Passwords are used to authenticate	Not-encrypted
Uses the network to log	0.1	0.2	0.7
Passwords are used to authenticate	0.2	0.1	0.7
Not-encrypted	0.7	0.7	0.1

Similarity Function f_2

Related Business Rule	Available over the internet	Restricted services	Restricted access
Available over the internet	1.0	0.1	0.1
Restricted services	0.1	1.0	0.8
Restricted access	0.1	0.8	1.0

Assumption	Uses the network to log	Passwords are used to authenticate	Not-encrypted
Uses the network to log	1.0	0.6	0.2
Passwords are used to authenticate	0.6	1.0	0.2
Not-encrypted	0.2	0.2	1.0

5 Conclusions

This paper introduces a system for capturing, sharing and reusing knowledge about mitigating or removing security flaws from a software system. Future directions include: experimenting the approach on a real case base, and improve the mechanisms of searching. With regards to the latter point, we will further investigate how to make more reliable the similarity function. As the comparison's techniques proposed here are somehow preliminary, we will take into account the application of regular expressions, string patterns, and heuristic based techniques. A second concern is about how to col-

Table 8. Misuse case #557.

Name	Tamper With DB
Summary	A crook manipulates the web query submitted from a search form, to update or delete information or to reveal information that should not be publicly available.
Date	2001.02.23
Author	David Jones
Basic Path	1. The crook provides some values to a product web form (e.g. the use case Register Account) and submits. 2. The system displays the result matching the query. 3. The crook alters the submitted URL, introducing an error in the query and resubmits the query. 4. The query fails and the system displays the database error message to the crook, revealing more about the database structure. 5. The crook further alters the query, for instance adding a nested query to reveal secret data or update or delete data, and submits. 6. The system executes the altered query, changing the database or revealing content that should have been secret.
Alternative Paths	ap1. In step 3 or 5, the crook does not alter the URL in the address window, but introduces errors or nested queries directly into form input fields.
Mitigation Points	mp1. In step 4, the exact database error message is not revealed to the client. This will not entirely prevent the misuse, but the crook will have a much harder time guessing table and field names in step 5. mp2. In step 6, the system does not execute the altered query because all queries submitted from forms are explicitly checked in accordance with what could be expected from that form. This prevents the misuse case.
Triggers	t1. Always true
Preconditions	The crook is able to search for products, either because this function is publicly available, or by having registered as a customer.
Mitigation Guarantee	crook is unable to access the database in an unauthorized manner through a publicly available web form (cf mp2).
Related Business Rules	The services of the e-shop shall be available to customers over the internet.
Stakeholder and Threats	st1. E-shop: Loss of data if deleted. Potential loss of revenue if customers are unable to Order Product, or if prices have been altered. Badwill resulting from this. st2. Customers: potentially losing money (at least temporarily) if crook has malignantly increased product prices. Unable to order if data lacking, wasting time.
Potential Misuser Profile	Skilled. Knowledge of databases and query language, at least able to understand published exploits on cracker web sites.

lect values which define the similarity tables. Proper processes to identify and validate them will be modelled and assessed with empirical investigation.

References

1. A. Raman and S. Muegge, An integrated approach to security in software development methodologies, Proc. of Canadian Conference on Electrical and Computer Engineering. 2008, pp. 002011-002014
2. C. Lai, Java Insecurity: Accounting for Subtleties That Can Compromise Code, IEEE Software, IEEE Computer Society, 2008, pp. 13-19
3. C. Riesbeck and R. Schank, Inside Case-Based Reasoning, Riesbeck/Schank, 1989
4. D. Ahmad and I. Arce, Vulnerability Bazaar, IEEE Security and Privacy, IEEE Computer Society, 2007, pp. 69-73
5. D. Byers and N. Shahmehri, Design of a Process for Software Security, Proc. of the The Second International Conference on Availability, Reliability and Security (ARES), IEEE Computer Society, 2007, pp. 301-309.
6. D. Xu and K. N. Kendall, Threat-Driven Modeling and erification of Secure Software Using Aspect-Oriented Petri Nets, IEEE Transactions on Software Engineering, IEEE Press, 2006, pp. 265-278

7. G. Sindre, A.L. Opdahl and G.F. Brevik, Generalization/Specialization as a Structuring Mechanism for Misuse Cases. In Proc. of 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), 2002
8. J. A. Ingalsbe, L. Kunimatsu, T. Baeten and N. R. Mead, Threat Modeling: Diving into the Deep End, IEEE Software, IEEE Computer Society Press, 2008, pp. 28-34
9. J. Steven and G. Peterson, Defining Misuse within the Development Process, IEEE Security and Privacy, IEEE Computer Society, 2006
10. L. Wang, E. Wong and D. Xu, A Threat Model Driven Approach for Security Testing, In Proc. of the Third International Workshop on Software Engineering for Secure Systems (International Conference on Software Engineering), IEEE Computer Society, 2007, p. 10
11. M. E. Johnson and E. Goetz,Embedding Information Security into the Organization, Security & Privacy, IEEE Computer Society, 2007, pp. 16-24
12. N. A. Malik, M. Y. Javed and U. Mahmud, Threat Modeling in Pervasive Computing Paradigm, In Proc. of New Technologies, Mobility and Security, 2008, pp. 28-34
13. S. Barnum and G. McGraw, Knowledge for Software Security, Security & Privacy, IEEE, 2005, pp. 74-78
14. X. Li, and K. He, A Unified Threat Model for Assessing Threat in Web Application. In Proc. of the 2008 International Conference on Information Security and Assurance (isa 2008), 2008, pp. 142-145
15. X. Liu and Z. Liu Evaluating Method of Security Threat Based on Attacking-Path Graph Model. In Proc. of International Conference on Computer Science and Software Engineering, 2008 , pp. 1127-1132
16. A. Stahl, and T. Gabel. Using Evolution Programs to Learn Local Similarity Measures. In Proc. of the 5th International Conference on Case-Based Reasoning (ICCBR 2003), Trondheim, Norway, June 2003.
17. D. Mellado, E. Fernández-Medina, and M. Piattini, "SREPPLine: Towards a Security Requirements Engineering Process for Software Product Lines". In Proc. of WOSIS 2007, 2007, pp. 220-232