Journal of Universal Computer Science, vol. 15, no. 15 (2009), 3059-3078
submitted: 1/2/09, accepted: 29/8/09, appeared: 1/9/09 © *J.UCS*

A System for Managing Security Knowledge using Case Based Reasoning and Misuse Cases

Corrado Aaron Visaggio
(University of Sannio, Benevento, Italy
visaggio@unisannio.it)

Francesca de Rosa
(University of Sannio, Benevento, Italy
derosa@unisannio.it)

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.

Keywords: Misuse case, Case Base Reasoning, Security Knowledge Management
Category: D.2.9

1 Introduction

Knowledge about software security is now acquiring an economic and strategic value for Organizations: since a decade, a market of vulnerabilities has been developing and expanding fast [Ahmad, 2007]. In order to improve security into software products, hiring skilled professionals or leveraging individual competencies and capability is not enough for successfully facing security concerns, according to Johnson and colleagues [Johnson, 2007].

As pointed out by Barnum and McGraw [Barnum, 2005], critical knowledge built during the usual problem solving activities concerning software security, should be captured and widely shared within an organization. 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 modelling is a central aspect of the security engineering process [Byers, 2007].

A way to model threats in terms of interaction with the system is the misuse case [Steven, 2006]. A misuse case describes potential system behaviours 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;
- technology 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 [Lai, 2008].

Of course, the misuse case could also exploit flaws concerning more than one level. The complexity of a security flaw is a source of project delay, cost increasing, and, generally, risks, sharing as much as possible the knowledge regarding these concerns is a good means to handle such a complexity and, consequently, reduce risks and damages due to the incapability of dealing properly with security issues.

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. When a process stakeholder needs to solve a security flaw at any phase of software process (analysis, design, code, test), she can submit a query to the knowledge base. The system finds those vulnerabilities which were successfully solved (and whose solution could be retrieved in the knowledge base) similar to the submitted 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 exclusively 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 of formalization of a misuse case. Section 4 shows exemplar working of the algorithm. Section 5 discusses related work. Finally, conclusions are drawn.

2 The proposed System

The system we propose in this paper aims at supporting the entire lifecycle of knowledge about the software security flaws within a software Organization. The knowledge lifecycle (see Figure 1) includes three phases:

1) *Knowledge creation*, during which knowledge is created. The basic assumption is that knowledge is created when a solution for a new problem is found and validated as working.
2) *Knowledge retention*, during which the new knowledge is embodied into the existing knowledge base, so that knowledge can be shared among process stakeholders.

3) *Knowledge usage*. The existing knowledge is used by the process stakeholders according to the needs which arise during the usual software process.

As we are interested in the knowledge necessary to remove a security flaw into the software system at any stage of development, a knowledge chunk is formalised as a couple (problem, solution), i.e. a case.

2.1 The Knowledge lifecycle

The knowledge base is a set of security problems that could be encountered into the development of a software system, paired with a suitable solution. Of course, the suitability of the solution is validated by the experience, i.e. by applying that solution when the correspondent problem arises and verifying that it works. The cases are not static, as the proposed solution could fail: the case should be improved or it is needed that a new case is created form scratch. Let's analyse each phase of the knowledge lifecycle.

2.1.1 Knowledge usage

The knowledge base has the main purpose of sharing the knowledge regarding strategies to adopt for solving a security flaw. The system becomes a repository of cases that any software process stakeholder could interrogate in order to obtain a solution for a specific problem at any phase of the software process. Knowledge usage is the very focus of this paper and it will be discussed in detail later. Roughly speaking, this phase starts with the definition of the problem. The problem is defined throughout a structured form that the process stakeholder could fill in by selecting from a list of existing key words.

The system searches for similar cases, i.e. the set of cases which solve a problem close to the one submitted by the process stakeholder. The similarity among two cases is a mathematical function calculated according to a specific algorithm, which is one of the key part of our system. An existing case could be completely used, i.e. the solution proposed is successfully applied in practice. Otherwise, if it does not fit the actual problem, a new case is created: this is the *knowledge creation* phase.

The usage of knowledge could also suggest an improvement initiative for an existing case as the definition of either the problem or the solution (or both) could be improper. This requires activities of *knowledge retention*.

2.1.2 Knowledge creation

A new case is created when a problem arises and there is not a case in the knowledge base that provides a suitable solution for that problem. This could happen for two reasons:

i) the problem was never encountered before, thus the solution does not exist in the knowledge base

ii) similar problems were encountered before, but the existing solutions do not fit the current problem. This happens because the problem has not a proper grain, and needs to be further detailed. In this case the existing solutions could be helpful to define the new solution. A new case is created, even if the problem is similar to existing ones.

2.1.3 Knowledge retention

When a new case is created it must be stored in the knowledge base. In order to maintain the knowledge base consistent and usable two main activities should be performed:

i) *remove redundant cases*. Redundant cases are due to two kinds of anomalies. The first anomaly occurs when two cases have the same formulation of the problem but different solutions, or vice versa, when the same definition of the solution is related to different problems. This could hide a flawed validation of the solution or the part in common – problem or solution- needs to be better detailed. We recall that this is not necessarily an anomaly, as a problem could have different solutions, or the same solution can fit well to different problems. In this situation the two cases must be merged in one. The second kind of anomaly occurs when two cases are substantially equals and redundant, even if formally different.

ii) *align the similarity relationships between cases*. Similarity among cases is a mathematical relationship which helps to establish which is the most suitable solution in the base for the problem submitted by the process stakeholder. Similarity calculation depends on weights defined by the base administrator. When new cases are created these weights could need to be properly changed in order to keep effective the similarity calculation. This aspect will be detailed in the following.

2.2 The Reasoning Model

With this paper we adapt the case base reasoning (CBR) [Riesbeck, 1989] mechanisms for capturing, sharing, and reusing knowledge about security threats within a Software Organization.

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 (Figure 2).

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 than validated (**revise phase**). Finally the new case must be integrated in the case base (**retain phase**).

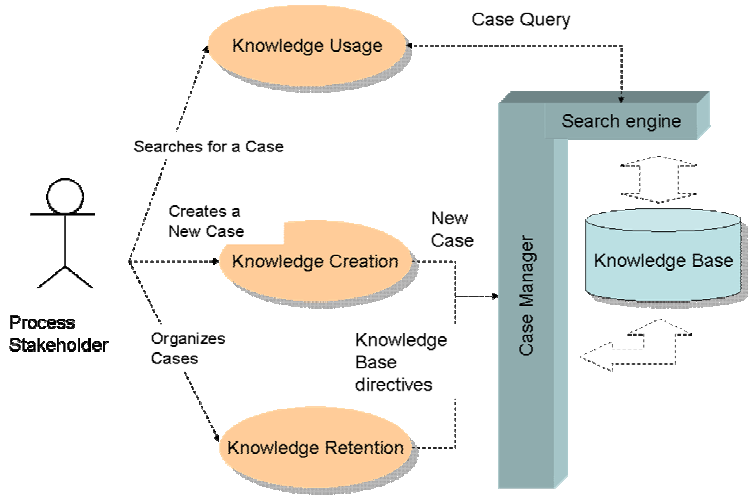

Figure 1: The main phases of the System

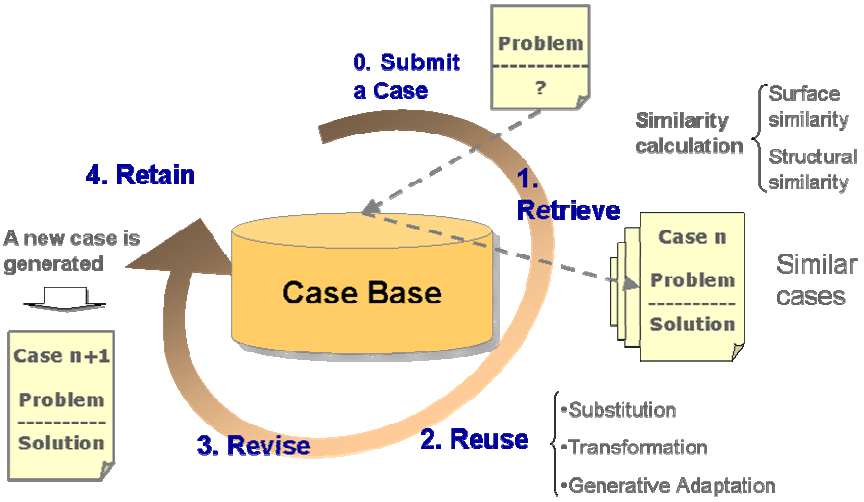

Figure 2: The Case Based Reasoning Process

2.3 The model of searching

In this section the model for searching the case which fits the encountered problem is introduced.

Let KB be the knowledge base KB and let O_j a case, than the knowledge base is a collection of cases:

KB= { $O_1, O_2, \ldots, O_j$},

Let $O_\alpha\{a_1, a_2, \ldots, a_n\}$ be a complex object with *n* attributes a_i, where a subset of these attributes represents the definition of the problem and the remaining set of attributes represents the solution.

The n attributes make the Knowledge Base an n-dimensions space; consequently we are able to define distances between each couple of objects so that any object has its own position into the knowledge base. The distance between two objects O_i, O_j is named *GlobalSim* O_α *(*O_i,O_j *).*

The possibility of identifying a position of an object in the objects' space will help us to properly retrieve the object we are looking for. As illustrated in Figure 3, if the problem is formalised in the object O_1, O_3's problem is most similar to O_1's than O_5's, as the distance is smaller.

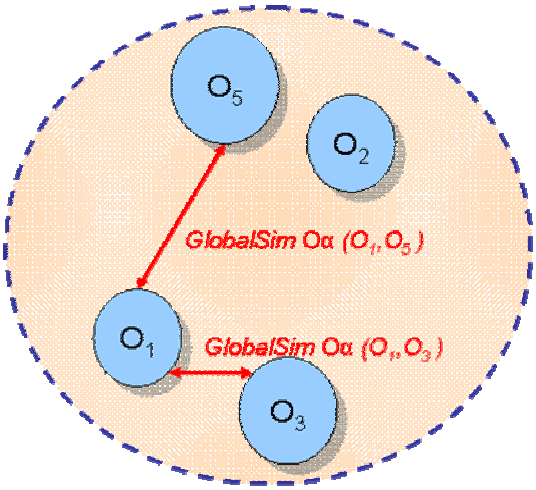

Figure 3: The Space of cases

The structure of the case refers to the specifications of misuse case provided by Sindre et al. [Sindre, 2002], 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 values must be used when the case is filled in. Such values, which are basically key words, are named **domain's tag**. The usage of domain's tag help for the calculation of similarity as explained later in the paper. 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.

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 Paths	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.
Stakeholder 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

Table 1: Structure of a misuse case

The attributes which define the solution are: Mitigation Points, and mitigation guarantee. The attributes which define the problem are all the remaining ones except for Name, Summary, Date, and Author.

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.

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

Table 2 : An exemplar problem

The exemplar problem consists of understanding how to mitigate the risk that passwords used to authenticate for restricted services are captured by other users or lost.

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 fulfils these properties: reflexivity, symmetry, monotony, and triangle equality. 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_α $(O_1,O_2) = \frac{1}{n}\sum_{i=1}^{n} localSimi$. The way of local similarity calculation depends of the kind of objects' attribute. In case of: numbers, similarity is a distance; strings, similarity is an evaluated comparison; symbols, similarity is calculate 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: $\frac{1}{n}\sum_{i=1}^{n} | goldStdi - simValuei |$, where n is the number of comparisons, $goldStd_i$ is the gold standard value and $simValue_i$ 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* [Stahl, 2003], which is a hyperbolic function must be defined as

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

$$a = \frac{fitnessMean * deviationMax}{fitnessMax - 2\,fitnessMean}$$

$$b = \frac{fitnessMax * fitnessMean}{fitnessMax - 2\,fitnessMean}$$

$$z = a * (fitnessMax + b)$$

- $deviation_{max}$, 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 *a* be an attribute and let a_i, with i ∈ [1,k] be a possible value assumed by *a*, being *a* ∈ A{ a_1, a_2, a_3, a_4,…,a_k}, and A the domain in which *a* varies. A similarity table, i.e. T_a, for the attribute *a* 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 ***Knowledge Usage*** (which recalls the ***CBR's retrieve phase***) phase 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 (CBR's ***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 (***Knowledge Creation***). 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. CBR's r***evise phase*** consists of verifying that the solution is effective. Finally, the case is catalogued in the case base (***Knowledge Retention)***. 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.

	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 3: Similarity table for "Stakeholders and Threats" attribute belonging to the similarity function f_2

3 An exemplar Case

In table 4 a misuse case is presented; it concerns the manipulation of the query submitted to a database from a web form. SQL injection is a typical technique exploited for this purpose.

The mitigation strategies suggested are two: the first one consists of hiding the error page, from which the attacker can infer knowledge about the database structure and to validate the input, in order to allow the execution only for the queries showing a proper formulation. Domain tags are typed as bold.

4 Exemplar working of the algorithm

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 4.

Name	Tamper With DB
Summary	A crook manipulates the web query submitted from a search form, to update ordelete 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 contentthat 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 toOrder Product, or if prices have been altered. Badwill resulting from this. st2. Customers: potentially losing money (at least temporarily) if crook has malignantlyincreased product prices. Unable to order if data lacking, wasting time.
Potential Misuser Profile	Skilled. **Knowledge of databases** and **Knowledge of query language**, at least able to understand published exploits on cracker web sites.

Table 4: Misuse Case #557

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 5. 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.

Similarity Function f_1 – *Related Business Rule*			
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
Similarity Function f_1 – *Assumption*			
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*			
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
Similarity Function f_2 – *Assumption*			
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

Table 5: Comparing similarity tables of functions f_1 and 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 6: Similarity table for "Stakeholders and Threats" attribute belonging to the similarity function 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

Table 7: Comparing retrieval results by applying the two similarity functions f_1 and f_2.

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 7) 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 8).

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 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.

4.1 An exploratory case study

With the following brief case study we will show how important is the definition of the similarity function for the success of the retrieve phase. Let's take into account the problem illustrated in table 10, regarding a real vulnerability of PDF files.

Our case base is populated with fourteen different cases, and we consider two different similarity function f_3 and f_4, whose details are not provided here. Results are depicted in table 11.

	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 loss of data; possible disclosure of data, possible alteration of data. May disrupt business and affect customer relations	*No Value*

Table 8: Comparing #530 and #541 misuse cases.

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 [..]

Table 9: Evaluating suitable of Retrieved Case #524.

Trigger	Always true
Preconditions	User is connected to Internet User uses Internet Explorer User opens PDF file with Adobe Acrobat or Acrobat Reader
Assumption	User has extended privileges User executes a vulnerable version of application
Mitigation Guarantees	Access to PDF file from trusted or known sources Prevent IE from automatically opening PDF documents
Related Business Rule	User may convert any PDF documents
Stakeholders and Threats	Loss of data User loses control over its PDF file denial-of-service

Table 10: Problem formulation of the case study

Misuse case ID	f_3		f_4	
	Fit. mean 0.05	**Fit. Mean 1.00**	**Fit. mean 0.05**	**Fit. mean 1.00**
560	16	27	19	30
562	12	23	28	39
563	17	28	26	37
564	20	31	18	28
565	34	34	25	25
566	27	27	47	47
567	32	32	39	44
568	18	26	21	28
569	17	27	17	27
570	31	31	31	31
571	16	24	25	34
572	17	25	31	40
573	38	45	22	42
574	33	41	32	32

Table 11: Comparing results generated by similarity functions f_3 and f_4.

According to f_3 the most similar case is #573, where as according f_4, the most similar case is #566. Let's compare the two cases (tables 12-13).

We observe that both the cases concern denial of services, but the two vulnerabilities reported into the two cases are very different. In the case of #573 the midi files vulnerability lets to execute malicious code, that is very close to our problem. Conversely, #566 deals with a flawed garbage collection mechanism, which is very different from the kind of problem we are trying to solve.

Name	Apple Quicktime fails to properly process specially crafted MIDI files
Summary	The Apple Quicktime player contains a heap buffer overflow vulnerability. This vulnerability may allow an attacker to execute arbitrary code or create a denial-of-service condition.
Date	06/03/2007
Author	Apple Computer, Inc.
Basic Path	1. Browser (on Mac OS X or Microsoft Windows/XP/Vista operative system) automatically opens a midi file using QuickTime (versions prior 7.1.5) without user interaction 2. Attacker triggers the overflow and execute arbitrary code
Alternative Paths	Ap1. In step 1, User opens a specially crafted midi file with QuickTime. The file is supplied on a web page, in an email from attacker Ap2. In step 2, Attacker triggers the overflow and create a denial-of-service Condition
Mitigation Points	Mp1. In step 1, User uses a QuickTime version later to 7.1.5 or he doesn't use Mac OS X or Microsoft Windows/XP/Vista as operative system Mp1. In step 1, Browser doesn't automatically open a midi file using QuickTime or user has a good antispam system Mp2. In step 2, Attacker can't trigger the overflow because user has reduced privileges.
Triggers	The crafted midi file is open using QuickTime
Preconditions	P1. User has extended privileges
Assumptions	P1. User accepts to open a (crafted) midi file using QuickTime or Browser automatically open this file without user interaction
Mitigation Guarantee	Apple has released an update to address this issue. Until updates can be applied, do not allow web browser to open files associate with QuickTime automatically. Do not open multimedia files that are from untrusted or unknown sources. Running QuickTime with reduced privileges may help mitigate the effects of this vulnerability.
Related Business Rules	User may use the system with reduced privileges.
Stakeholder and Threats	1. Loss of data 2. Impossible to access to own system
Potential Misuser Profile	Misuser is able to write and use an exploit code

Table 12: Misuse case #573.

This exploratory case study suggests that the correct formulation of the similarity functions is very important for a successful retrieval of cases.

Name	Mozilla Firefox JavaScript engine fails to properly handle garbage collection
Summary	Mozilla Firefox JavaScript engine fails to properly handle garbage collection. This vulnerability result in memory corruption. A remote, unauthenticated attacker may be able to cause a vulnerable version of the Firefox browser crash
Date	18/04/008
Author	Mozzilla
Basic Path	1. User opens a vulnerable version of Firefox browser (version prior 2.0.0.14) 2. Attacker exploits the vulnerability, to crash the Firefox application
Alternative Paths	Ap1. In step 1, User open a vulnerable version of Thunderbird (prior 2.0.0.24) or SeaMonkey (prior 1.1.10)
Triggers	Always
Preconditions	1. User is connected to Internet
Assumptions	1. User executes a vulnerable version of Firefox application
Mitigation Guarantee	User has to update to Firefox 2.0.0.14, Thunderbird 2.0.0.14, or SeaMonkey 1.1.10. Using th Mozilla Firefox NoScript extension to whitelist web sites that can run scripts and access installed plugIns will mitigate this vulnerability.
Stakeholder and Threats	User is unable to use Firefox application Misuser causes the crash of Firefox application
Potential Misuser Profile	Misurer is able to exploit the Firefox vulnerability

Table 13: Misuse case #566.

5 Related Work

At the best knowledge of the authors the problem of capturing and reusing security knowledge modelled as misuse case has been not faced. Ingalsbe et al. [Ingalsbe, 2008] introduce a process of threat modelling basically aimed at risk mitigation. Modelling the threats is used as a basis for evaluating related risks. This paper copes with the organizational aspects of threat modelling. Some authors [Raman, 2008] 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 [Li, 2008] 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 [Liu, 2008] 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 [Malik, 2008] 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 [Xu, 2006] presents a formal threat-driven approach, which explores explicit behaviours 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 modelling. Paper [Wang, 2007] proposes a threat model-driven security testing approach for detecting undesirable threat behaviour at runtime. 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 modelling for strengthening security testing. Matulevičius et al. [Matulevicius, 2008] analyse how to improve the misuse case in order to better support the risk management activities. Their research is aimed at integrating misuse case into the software analysis phase rather than use them for sharing security knowledge within organizations. Whittle and colleagues [Whittle, 2008] merge three different techniques in order to implement an executable modelling system for misuse case, that allows users to animate misuse case into the related use case. This system is mainly oriented to the design and testing phase. Pauli and Xu [Pauli, 2006] model functional requirements with use cases and use the STRIDE threat categories from the threat modelling approach to identify misuse cases. The interplay between misuse cases and use cases drive the identification of mitigation use cases that preserve the goals of security. These cases are then systematically decomposed to allow the details of each case to be specified for the benefit of security requirements. Okubo and Tanaka [Okubo, 2008] extend misuse case description with fine classification of mis-actors, additional definition of data asset elements and fine classification of misuse case endpoints. Saleh and Habil [Saleh, 2008] propose a model to elicit threats mitigation trough the design of systems and analysis of misuse case. In summary, misuse cases are used mainly for supporting design and testing phase in the software process, rather than for capturing knowledge.

6 Conclusions and Future Work

On the one hand security is becoming a critical quality factor of software systems. On the other hand, the increasing complexity of software technology makes really hard to elaborate successful solutions to security flaw.

This situation could be faced by sharing within software organizations the knowledge about software security built during the mitigation or removal of a security flaw. The system uses the model of case based reasoning for managing the lifecycle of knowledge, that is formalized as set of misuse cases.

In the near future we plan to improve the form of similarity function in order to understand which is the one that fits properly the search needs.

References

[Ahmad, 2007] Ahmad, D., Arce, I., Vulnerability Bazaar, IEEE Security and Privacy, IEEE Computer Society, 2007, pp. 69-73.

[Barnum, 2005] Barnum, S., McGraw, G., Knowledge for Software Security, Security & Privacy, IEEE, 2005, pp. 74-78.

[Byers, 2007] Byers, D., Shahmehri, N., Design of a Process for Software Security, in Proc. of the The Second International Conference on Availability, Reliability and Security (ARES), IEEE Computer Society, 2007, pp. 301-309.

[Ingalsbe, 2008] Ingalsbe, J. A., Kunimatsu, L., Baeten, T., Mead, N. R., Threat Modeling: Diving into the Deep End, IEEE Software, IEEE Computer Society Press, 2008, pp. 28-34.

[Johnson, 2007] Johnson, M. E. , Goetz, E., Embedding Information Security into the Organization, Security & Privacy, IEEE Computer Society, 2007, pp. 16-24.

[Lai, 2008] Lai, C. Java Insecurity: Accounting for Subtleties That Can Compromise Code, IEEE Software, IEEE Computer Society, 2008, pp. 13-19.

[Li, 2008] Li, X. , He, K. ,A Unified Threat Model for Assessing Threat in Web Applications, Proceedings of the 2008 International Conference on Information Security and Assurance (isa 2008), IEEE Computer Society, 2008, pp. 142-145.

[Liu, 2008] Liu, X., Liu, Z. , Evaluating Method of Security Threat Based on Attacking-Path Graph Model, Computer Science and Software Engineering, 2008 International Conference on, 2008, pp. 1127-1132.

[Malik, 2008] Malik, N. A., Javed, M. Y., Mahmud, U., Threat Modeling in Pervasive Computing Paradigm, New Technologies, Mobility and Security, 2008. NTMS '08, 2008, pp. 28-34.

[Matulevičius, 2008] Matulevičius, R., Mayer, N., Heymans, P., Alignment of Misuse Cases with Security Risk Management, Proceedings of The Third International Conference on Availability, Reliability and Security, 2008, IEEE Computer Society, pp. 1399-1404.

[Okubo, 2008] Okubo, T., Tanaka, H., Identifying Security Aspects in Early Development Stages, proc.of The Third International Conference on Availability, Reliability and Security 2008, IEEE Computer Society, pp. 1150-1155.

[Pauli, 2006] Pauli, J., Xu, D., Integrating Functional and Security Requirements with Use Case Decomposition, Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06), 2006.

[Raman, 2008] Raman, A., Muegge, S., An integrated approach to security in software development methodologies, in Proceedings of Canadian Conference on Electrical and Computer Engineering. 2008, pp. 002011-002014.

[Riesbeck, 1989] Riesbeck, C., Schank, R., Inside Case-Based Reasoning, Riesbeck/Schank, 1989.

[Saleh, 2008] Saleh, K., Kabhil, M., The Security Requirements Behavior Model for Trustworthy Software, Proceedings of 2008 International MCETECH Conference on e-Technologies, 2008, IEEE Computer Society, 235:238.

[Sindre, 2002] Sindre, G. , Opdahl, A.L. , Brevik, G.F. , Generalization/Specialization as a Structuring Mechanism for Misuse Cases, 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), 2002.

[Stahl, 2003] Stahl, A., Gabel, T. , Using Evolution Programs to Learn Local Similarity Measures, in Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR 2003), Trondheim, Norway, June 2003.

[Steven, 2006] Steven, J. , Peterson, G. Defining Misuse within the Development Process, IEEE Security and Privacy, IEEE Computer Society, 2006.

[Xu, 2006] Xu, D., Kendall, K. N. Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets, IEEE Transactions on Software Engineering, IEEE Press, 2006, pp. 265-278.

[Wang, 2007] Wang, L. , Wong, E., Xu, D. , A Threat Model Driven Approach for Security Testing, Proceedings of the Third International Workshop on Software Engineering for Secure Systems (International Conference on Software Engineering), IEEE Computer Society, 2007, p. 10.

[Whittle, 2008] Whittle, J., Wijesekra, D., Hartong, M., Executable Misuse Case for modelling Security Concerns, proc. of Int'l Conference on Software Engineering, 2008, IEEE Computer Society, 121-130.