Refining the evaluation of the degree of security of a system built using security patterns

Olga Villagrán-Velasco
Dept. of CS and Eng.
Universidad Nac. Autonoma de Mexico
Mexico
ovvingtel@gmail.com

Eduardo B. Fernández
Dept. of CS and EE
Florida Atlantic University
USA
fernande@fau.edu

Jorge Ortega-Arjona
Dept. of CS and Eng.
Universidad Nac. Autonoma de Mexico
Mexico
jloa@ciencias.unam.mx

Abstract

Evaluating the degree of security of a specific software system is a difficult problem and many metrics have been proposed. However, if the system has been built with a methodology that uses patterns as artifacts, a systematic and rather simple evaluation is possible and a metric has been proposed for this evaluation: perform threat enumeration, check if the patterns in the system can stop the identified threats, and calculate the coverage of these threats by the patterns. We refine here that approach by considering the additional effect of the policies (requirements) defined for the system and by using weights for threats and policies.

1 Introduction

A frequent practical problem is verifying if a system built using some either a systematic methodology or ad hoc ways, has reached some degree of security. Although there are many proposed metrics, there are few widely accepted metrics for security, and they are not easy to apply. This makes it difficult to compare the products of specific methodologies, to improve the security of a specific system, or to improve the methodology used to develop the product.

By security we mean the ability of a system to protect the assets in its applications against attacks from external and internal attackers. Security implies the provision of confidentiality, integrity, availability, and accountability. Some work tries to prove that systems have these properties, usually applying formal methods. We consider looking at how threats are handled as a more practical approach and in this work we take this orientation. We are not interested either in code-based measures, which do not really measure the whole system security. Security is a difficult problem, especially for complex applications, which have many avenues for attacks that include all the architectural levels of the system and where interactions between units may hide threats.

In this work we consider systems designed using *patterns*. A pattern describes a solution to a recurrent software or systems problem in a given context, and when the problems are security problems we call them *security patterns*. Security patterns provide a way for guiding system designers who are not experts on security to build secure systems [6]. Good security design requires the application of a set of principles [14], and the use of patterns is a convenient way to implicitly apply security principles even by people having little experience and/or little security knowledge. There have been some attempts to build catalogs of security patterns such as [6] and [17]. There are also several secure systems development methodologies that use patterns [6, 19, 20].

As a way to get a handle on the problem of security evaluation, Ref. [18] has suggested designing systems that exhibit measurable properties. Ref. [7] followed this idea and proposed a way to perform this evaluation. They first enumerate threats and verify if they all have been stopped or mitigated by some security mechanism realizing a security pattern. The percentage of threat coverage by patterns is their security measure. We show here how this approach can be refined by considering the effect of the policies (which are in effect additional security requirements) defined for the system and by using estimated weights for threats and policies according to their impact. We enumerate threats by considering all the actions in each use case and analyze how they could be subverted by attackers to reach their goals [6], but other threat enumeration methods are also acceptable.

We make clear that we are not defining a new methodology to build secure systems and we are not evaluating the effectiveness of a particular methodology to produce secure systems, we just evaluate the degree of security of a system already built or being built. In earlier work we have proposed one of those methodologies [20].

Our contributions include:

- The refinement of a proposed metric for evaluating system security based on threat enumeration and on verifying if these threats are or not controlled in a specific software architecture. The refinement is based on considering the effect of policies and the use of weights according to their impact. This metric applies to systems built using security patterns.
- An evaluation of this metric by analysis and an example.

Section 2 presents background, while Section 3 introduces the refined security metric. Section 4 considers threat enumeration. Section 5 shows the calculation of the new metric, including an example. Section 6 evaluates the refined metric. Section 7 discusses related work. We end with conclusions in Section 8. An appendix describes the full model of the example system.

2. Patterns and security development methodologies

A pattern is a solution to a recurrent problem in a given context [2]. Patterns are described using a template composed of a set of structured sections. A problem section describes a general problem and forces that constrain and define guidelines for the solution. The solution is usually expressed using UML class, sequence, state, and activity diagrams (although we usually don't need all these models). A set of consequences indicate what is the effect of the pattern and how well the forces were satisfied by the solution, including advantages and disadvantages of using the pattern. An implementation section provides hints on how to use the pattern in an application, indicating what steps are needed and possible realizations. A section on "Known uses" lists real systems where this solution has been used previously. A section on related patterns indicates other patterns that complement the pattern or that provide alternative solutions. A pattern embodies the knowledge and experience of software developers and can be reused in new applications; carefully-designed patterns implicitly apply good design principles [14]. Patterns are also good for communication between designers and to evaluate and reengineer existing systems. While initially developed for software, patterns can describe hardware, physical entities, and combinations of these, as well as non-technical processes such as teaching a course or organizing a conference. In particular, security patterns can suggest solutions to designers who don't have much security experience. Because of their abstraction properties, security patterns provide a way to apply a holistic approach to system security and they are useful to handle large and complex systems in a comprehensive and unified way. To be effective a catalog is needed and a few exist [6, 17].

A security methodology SM can be defined formally as a couple: SM (SP; CF), where SP is a security process – the activities and/or steps taken to secure a software system of some type; and CF is a conceptual security framework, consisting of the conceptual artifacts used by the methodology's process, that must include a set of security solutions (defensive artifacts), as well as a set of threats (offensive artifacts) [20]. The survey in [19] identified several methodologies that use security patterns as main artifacts.

3. A refined security metric

Ref. [7] proposed a metric as the quotient of the number of threats controlled by the system patterns over the total number of threats. We now try to make it more precise by adding the effect of institution policies. We assume the existence of system documentation indicating the security and policy patterns that have been used in building the system.

Policies include general enterprise policies, security policies, security regulations, and industry standards. These policies are additional requirements and can be represented as *Requirement patterns* (RPs), which have a similar structure as security patterns. RPs can have three priorities:

Low (value 1). Satisfying this policy is desirable.
Medium (value 2). Satisfying this policy is important.
High (value 3). This policy must be satisfied.

We use as example a financial institution described in the Appendix, its use cases are shown in Fig. 4. The policies for the use case "Check Trade Information" (Fig. 1) could be:

RP1: All the sessions initiated by the Auditor must be logged (low priority)

RP2: All the actions performed by the Auditor during a session must be logged (high priority)

RP3: An auditor can only inspect the orders assigned to her (high priority).

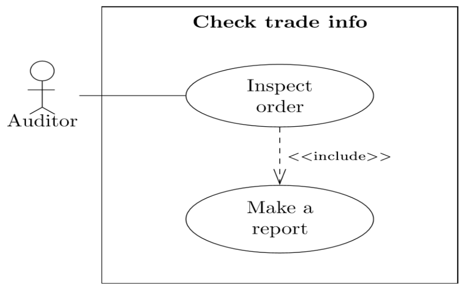

Fig. 1. Use case "Check Trade Information"

We use a UML activity diagram to describe the activities in a use case (Figure 2).
For this use case its security patterns (see Fig.5) could be:
Pat1: RBAC
Pat2: Authenticator
Pat3: Security Logger / Auditor

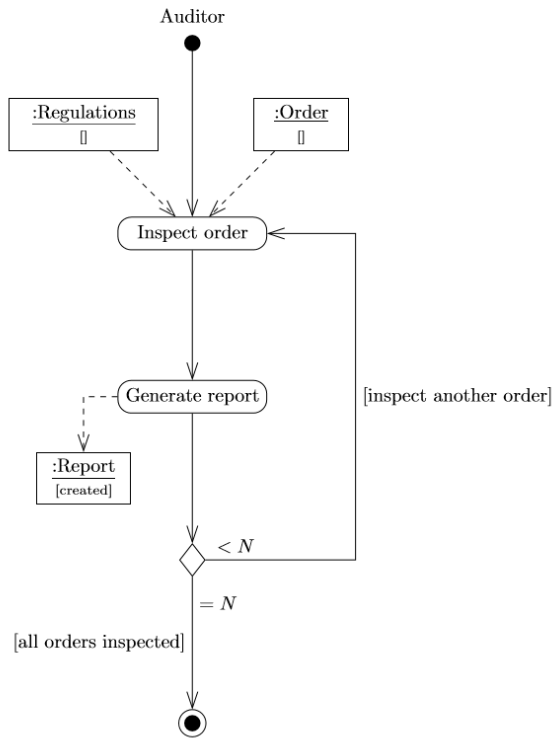

Fig. 2. Activity diagram of use case "Audit Trade orders"

4. Enumerating threats

This process is usually performed during the requirements and the design stages of the software development cycle and it analyzes each activity in the activity diagram of a use case to see how it could be subverted by an attacker to reach her goals [1]. The process requires to consider the activities in the use cases of the complete system. If this process has not been done in the system under analysis we can perform it as part of this evaluation. We show an example in the next paragraph.

Threat enumeration is a basic step in any secure system development methodology. This analysis results in a set of threats and since the use cases are all the ways to interact with a system we can enumerate threats systematically (although we cannot prove that we have found all of them). Secure development methodologies then consider which policies can stop or mitigate these threats and realize the policies with patterns. This process requires developers to conjecture possible attacks to different assets or parts of a system, to assess their impact and likelihood, and to determine how they could potentially be stopped or mitigated.

Because of the large number of threats that may appear as attacker goals, many of which may not be significant, it is important to reduce their number by performing a risk analysis. In this process threats must be ranked by impact and filtered or given weights before applying them in the calculation of SC. Note that the OWASP or CVSS scores are not useful here because they only consider design or code aspects. This means that designers must estimate the impact of threats. A simple approach is to assign three levels to threats:

Low. The threat has a low impact on the institution (1)
Medium. The threat has a significant impact in the institution (2).
High. The threat has a serious impact on the institution (3).

These criteria let us give a weight to each threat. A threat may have a higher impact in one system with respect to another depending on its context; e.g., getting the information of a bank customer is different from getting the tweets of a participant in a social network. The fact that the goals of the attacker are defined at a higher level than design aspects make these impacts easier to estimate.

Fig. 3 shows the threats for the use case "Audit Trade Orders in a financial institution". For the actor Auditor we can identify the following threats:

Inspect Order:
T11 Low impact. Deny to have inspected an order
T12 High impact. Copy information from orders
Generate Report:
T21 High impact . Ignore the policies that should have been applied in an order
T22 High impact. Illegal Dissemination of information from reports
T23 Medium Impact. Read information from other reports.

5. Calculating the new metric

5.1 Effect of threats

Each threat has a weight defined by:

$$\alpha = \frac{imp}{M}$$

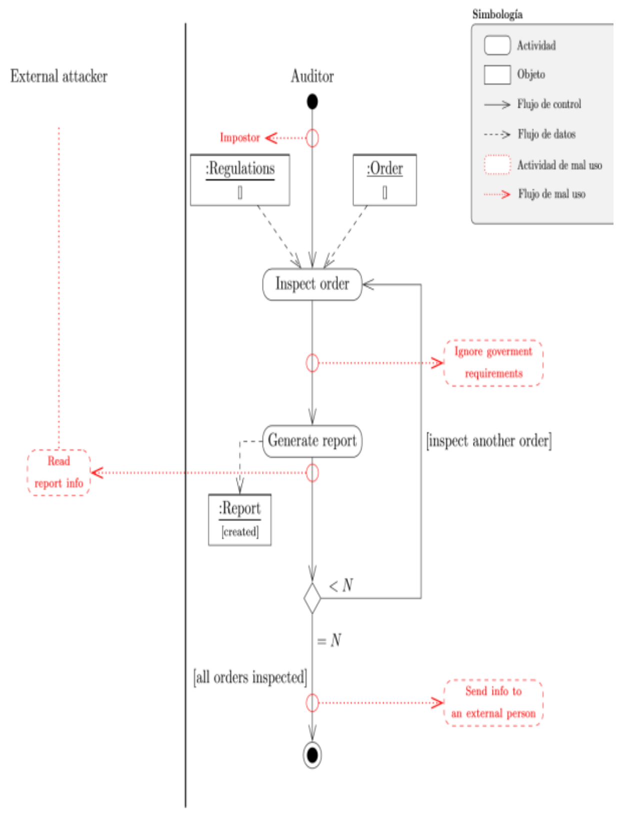

Figure 3. Activity Diagram for Use Case "Audit Trade Orders" showing threats as attacker goals.

Where α is the weight of the threat, imp is the impact of the threat, and M is the total number of identified threats. To calculate the weight of all the mitigated threats we have:

$$w_{ame} = \frac{\sum_{i=1}^{M} \alpha_i \cdot v_{p_i}}{\sum_{i=1}^{M} \alpha_i} \qquad (0 \leq w_{ame} \leq 1)$$

Where ω_{ame} is the mitigated weight of the system, α is the weight of each threat, and v_{pi} is the pattern value for this threat (1 if it can control it).

As an example, Table 1 shows the calculation for the use case shown earlier. We can see that since this system had a defense (security pattern) for all its identified threats its degree of security is 1. The patterns required here to stop these threats are: Pat1=Authorizer (RBAC), Pat2= Authenticator, Pat3=Security Logger/Auditor.

5.2 Effect of requirements

If there is a pattern that satisfies the policy we assign a value v =1; if not v = 0.

Table 1 Effect of threats on security

Threat	**Pattern(s)**	α	v_p	w_{ame}
T_{1_1}	Pat_3	$\frac{1}{5}$	1	
T_{1_2}	Pat_1	$\frac{3}{5}$	1	
T_{2_1}	Pat_3	$\frac{3}{5}$	1	$\frac{\frac{1}{5}\cdot1+\frac{3}{5}\cdot1+\frac{3}{5}\cdot1+\frac{3}{5}\cdot1+\frac{2}{5}\cdot1}{\frac{12}{5}} = 1$
T_{2_2}	Pat_1	$\frac{3}{5}$	1	
T_{2_3}	Pat_2	$\frac{2}{5}$	1	

Each policy (security requirement) has a system priority of:

$$\mu = \frac{prio}{N}$$

Where μ measures the importance of this policy in the system, *prio* is its priority, and N is the total number of policies defined for the system.

The weight of all the security policies is then:

$$w_{req} = \frac{\sum_{j=1}^{N} \mu_j \cdot v_{p_j}}{\sum_{j=1}^{N} \mu_j} \qquad (0 \leq w_{req} \leq 1)$$

Where ω_{req} is the weight of all the policies (requirements) in the system, μ_j is the importance of each policy, and v_{pj} is the value assigned to the presence of the pattern corresponding to each μ_j. Table 2 shows the result of this calculation for the running example.

Again, the result is 1 because we satisfied all the prescribed policies using the security patterns in the final system. The total degree of security of the system (ss) is then:

$$ss = w_{ame} \cdot w_{req}$$

Table 2. Satisfied requirements

Requirement	**Pattern(s)**	μ	v_p	w_{req}
Req_1	Pat_3	$\frac{1}{4}$	1	$\frac{\frac{1}{4}\cdot1+\frac{2}{4}\cdot1+\frac{1}{4}\cdot1+\frac{2}{4}\cdot1}{\frac{6}{4}}=1$
Req_2	Pat_3	$\frac{2}{4}$	1	
Req_3	Pat_1	$\frac{1}{4}$	1	
Req_4	Pat_1	$\frac{2}{4}$	1	

This metric combines the effect of handling the identified threats of the system (ω_{ame}) with the degree of fulfilling the institution requirements (ω_{req}). The metric ss takes values between 0 and 1. Clearly, we want the system to be close to 1. Depending on the type of application, we can accept values relatively low, say 0.5. In critical or important applications we need to be more strict and require values close to 1. Since we assume that we have access to the documentation of the system we can analyze the system design and see where new patterns need to be added to make the system stronger.

6. Analysis of the new metric

Savola presented criteria to evaluate the quality of a security metric [16], they were used in [7] and we also apply them to our metric. He found four basic quality criteria for these metrics:

- *Correctness*—threats are a basic quality criteria for security, mitigating them will improve the security of the system. Satisfying the requirements will also improve its security.
- *Measurability*—we can produce a reasonably complete list of threats and count the corresponding security patterns that can cover them. Counting the requirements is trivial because they are explicit.
- *Meaningfulness*—applying defenses against the identified threats will make the system more secure. Two sub-aspects of meaningfulness are:
  - *Comparability*—It is now possible to compare two systems based on their threat and policy coverage because we can obtain numerical values.
  - *Progression.* Adding security patterns we can improve security. Each added security or requirement pattern may improve security.
- *Usability*—the method for threat enumeration and the analysis of the use of patterns are rather simple approaches that do not require designers to be security experts.

Since we are looking at models we cannot say much about code vulnerabilities. However, we claim that with wise use of compartmentalization we can build systems where an attacker can get some data from a compromised section but not reach more valuable information. We cannot perform actual measurements to evaluate a design either, we can only indicate that specific threats cannot happen; that is, meaningfulness is high. These measures can be applied both to systems under development or to systems already built. In the second case we need to identify in the system the patterns that have been applied in its construction, either explicitly or implicitly. Applying these measures while a system is being built can help the designers selecting what patterns they need to add. If the process is iterative the designers can try different defenses to improve the metrics.

7. Related work

There is a variety of proposed measures for security. We discuss the most relevant to our work, starting with surveys. [12] is a survey of security metrics describing the state of the art up to 2010. They emphasize standards and indicate that security metrics are not very developed and need more work. Another survey [10] considers aspects of security measurement and possible research areas; it summarizes the state of the art up to 2009. [15] and [18] discuss measuring security in general; [18] classifies measures into computational-complexity metrics, economical/biological metrics, and empirical metrics. An important direction is represented by the concept of attack surface [11]. This measure counts the ways through which an adversary can penetrate the system. It does not consider the semantics of the application or the quality of the design, only its input/output properties.

Similarly to us, [9] also takes advantage of patterns to evaluate security. It associates security metrics to patterns and aggregates the measures of a system to evaluate its security. Their metrics are statistical measures based on system events, which implies the need to measure the actual system behavior. [8] quantifies the security level of a system based on its implemented/missing security patterns but their method uses a totally different (and more complex) approach.

There are several approaches to measure vulnerabilities in a specific system, e.g., [4, 5]. They only can measure a specific system implementation, they are not a measure of the quality of the design but they can complement our metrics by indicating where a pattern can be added to remove a vulnerability.

[13] proposes a security metric based on arguments. The metric relies on its degree of confidence in security arguments supporting a security goal. Confidence is based on appropriateness, sufficiency, and trustworthiness. It uses the CWE lists [5] to indicate where to apply the arguments. The problem is that a system may have many vulnerabilities and each one requires an argument.

The Common Criteria (CC) approach evaluates specific products according to protection profiles that define their expected requirements [3]. The Common Criteria provides assurance that the process of specification, implementation and evaluation of a product has been conducted in a rigorous, standard, and repeatable manner at a level that is commensurate with the target environment for use; however, it does not try to measure the degree of security of the evaluated product or the security of all the products of some methodology.

Some secure systems methodologies use security attributes as objectives and they use formal methods to prove that a system has a given degree of confidentiality or integrity. However, they often require unrealistic assumptions and do not consider implementation aspects. Their standard definition of security is the provision of properties such as confidentiality, integrity, availability, and accountability. However, these attributes are not directly measurable and proving that a system exhibits these properties is a very difficult problem for large and complex systems. We believe that a practical measure of security must be based on considering the threats to the system. In this case, instead of looking for abstract properties we need to find ways to stop the threats we have identified. The quality of the development methodology obviously has an effect on the security of its products; a set of criteria to evaluate this quality is given in [21].

8. Conclusions

We have presented a security metric which is simpler than earlier proposals and refines a similar type of method. While we require that the system has been built using patterns, even if the product software was not implemented using object-oriented methods it is still possible to define use cases (complete user interactions with the product; e.g. open account). Then each activity in a use case can be analyzed to see what are the goals of the attacker and his threats. Further, if no patterns were used in building a product, it is possible to discover them as abstractions of the security mechanisms that are actually implemented in the product.

Since a design can be implemented in many ways, this security evaluation applies to all its possible implementations. However, we are evaluating the design of a specific system, by showing that it has the correct patterns; however, it is possible that the patterns were not applied correctly. This can only be verified by analyzing or testing the code. Also, it is possible that although we have the correct patterns correctly implemented, there are still vulnerabilities in the code. An attacker, in this case could compromise parts of the system but not the complete system because a design based on patterns can provide a strong design structure. An interesting possibility is to consider the effect of safety patterns to define a metric useful for cyber-physical systems.

Further validation of these ideas requires applying them to a real system. We leave this as a future work.

References

[1] F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07). Turin, Italy, September 1-5, 328-333.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns, Volume 1. J. Wiley, 1996.

[3] Common Criteria Portal, last accessed February 22, 2015. https://www.commoncriteriaportal.org/

[4] Common Vulnerabilities and Exposures, https://cve.mitre.org/

[5] Common Weakness Enumeration (CWE), https://cwe.mitre.org

[6] E.B.Fernandez, "Security patterns in practice: Building secure architectures using software patterns". Wiley Series on Software Design Patterns. 2013

[7] E.B.Fernandez, N. Yoshioka, H. Washizaki, "Evaluating the degree of security of a system built using security patterns", 13th International Conference on Availability, Reliability and Security (ARES 2018), Hamburg, Germany, Sept. 2018

[8] S.T. Halkidis, N. Tsantalis, "Architectural risk analysis of software systems based on security patterns", IEEE Trans. On Dependable and Secure Computing, vol. 8, No 3, July-Sept. 2006, 129-142.

[9] T Heyman, R Scandariato, C Huygens, W Joosen, "Using security patterns to combine security metrics", Availability, Reliability and Security, 2008. ARES 08.

[10] W. Jansen, Directions in Security Metrics Research, NISTIR 7564, April 2009. http://csrc.nist.gov/publications/nistir/ir7564/nistir-7564_metrics-research.pdf

[11] K. Manadhata, J.M.Wing, "An attack surface metric", IEEE Trans. on Soft. Eng., vol. 37, No 3, May/June 2011, 371-386.

[12] D Mellado, E. Fernández-Medina, M Piattini, A comparison of software design security metrics, Procs. of the Fourth European Conference on Software Architecture, 2010, 236-242

[13] B.D.Rodes, J.C.Knight, K.S.Wasson, "A security metric based on security arguments", WETSoM'14, June 2014, Hyderabad, India, 66-72.

[14] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems", Procs. of the IEEE, vol. 63, No 9, Sept.1975, 1278-1308.

[15]W.H.Sanders, "Quantitative security metrics: Unattainable Holy Grail or a vital breakthrough within our reach", IEEE Security&Privacy, March/April 2014, 67-69.

[16] Reijo M. Savola: "Quality of security metrics and measurements". Computers & Security 37: 78-90 (2013)

[17] C. Steel, R. Nagappan, and R. Lai, Core Security Patterns: Best Strategies for J2EE, Web Services, and Identity Management, Prentice Hall, Upper Saddle River, New Jersey, 2005.

[18] Sal Stolfo, Steven M. Bellovin, and David Evans."Measuring security". IEEE Security & Privacy, 9(3):88, May--June 2011.

[19] A.V. Uzunov, E.B. Fernandez & K. Falkner (2012), "Securing distributed systems using patterns: A survey", Computers & Security, 31(5), 681 - 703. doi:10.1016/j.cose.2012.04.005

[20] Anton Uzunov, E. B Fernandez, Katrina Falkner, "ASE: A Comprehensive Pattern- Driven Security Methodology for Distributed Systems", J. of Comp. Standards & Interfaces, Vol. 41, Sept. 2015, 112-13

[21] Anton Uzunov, E.B.Fernandez, Katrina Falkbeer, "Assessing and Improving the Quality of Security Methodologies for Distributed

Systems", J. of Softw.Evolution and Process, August 2018, https://doi.org/10.1002/smr.1980

Appendix

We use a financial institution as a running example. Figure 4 shows its main use cases, while Figure 5 shows its class diagram. This model represents the facts that Customer have Accounts in which they can perform Transactions. There are two types of Customers, Owners are the entities responsible for the accounts, AccountUsers are the operational users of the accounts. The class diagram indicates in blue the security patterns that were added in order to counter the identified threats (Using the approach of [20]). Auditors prepare reports after inspecting Orders.

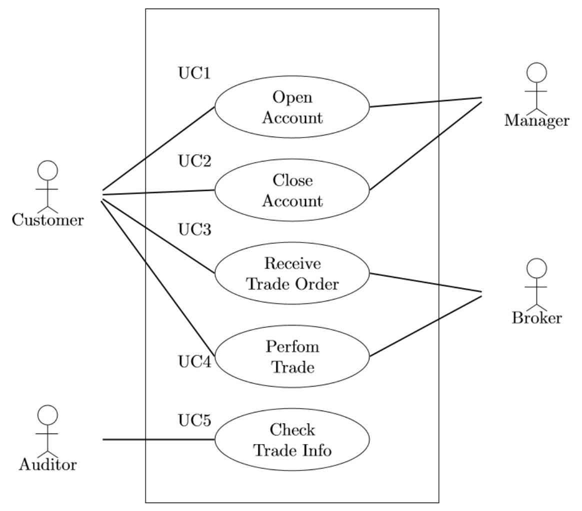

Fig.4 Use case diagram for a financial institution.

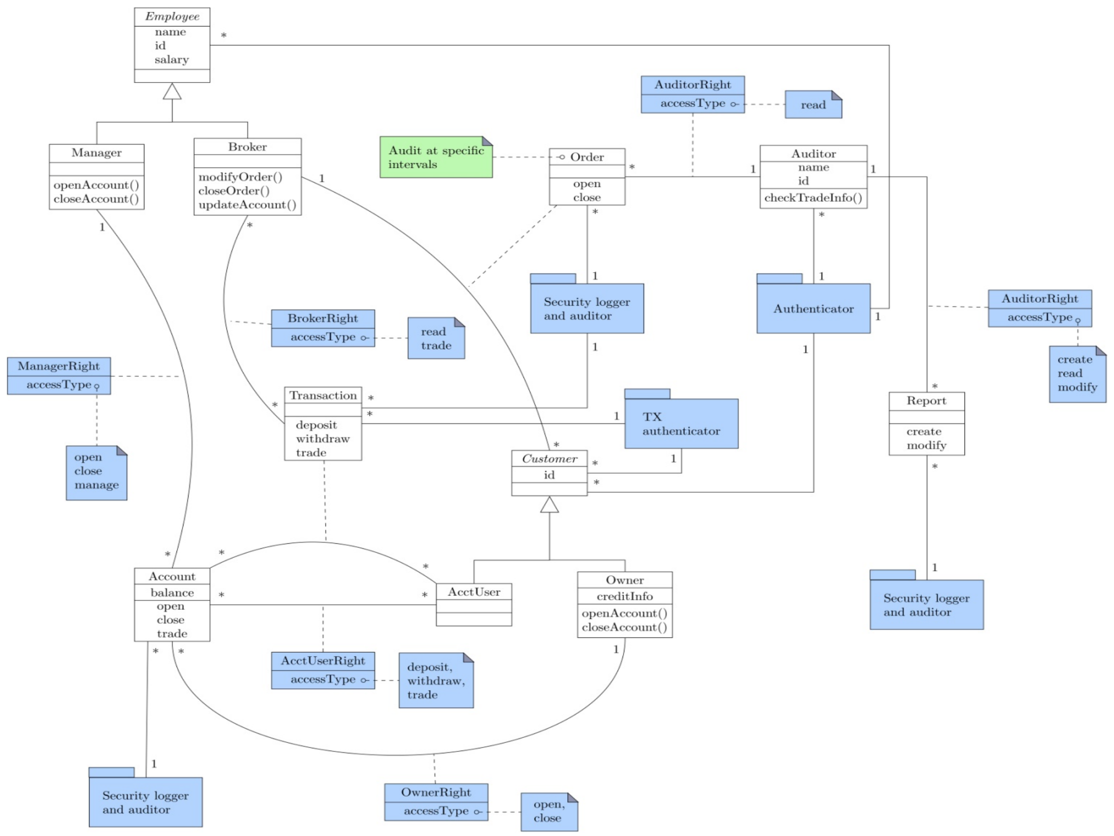

Fig. 5. Class diagram including security patterns for the financial institution (the patterns are described in [6])