A Security Requirements Approach for Web Systems

Article · January 2009

CITATIONS
12

READS
209

4 authors:

Stefan Wagner

273 PUBLICATIONS **5,099** CITATIONS

SEE PROFILE

Daniel Méndez Fernández

Blekinge Institute of Technology

217 PUBLICATIONS **2,926** CITATIONS

SEE PROFILE

Shareeful Islam

University of East London

71 PUBLICATIONS **1,770** CITATIONS

SEE PROFILE

Klaus Lochmann

IBM Deutschland GmbH

6 PUBLICATIONS **161** CITATIONS

SEE PROFILE

A Security Requirements Approach for Web Systems*

Stefan Wagner, Daniel Mendez Fernandez, Shareeful Islam, and Klaus Lochmann

Technische Universität München
Software & Systems Engineering
{wagnerst,mendezfe,islam,lochmann}@in.tum.de

Abstract. In order to avoid the high impacts of software vulnerabilities, it is necessary to specify security requirements early in the development on a detailed level. Current approaches for security requirements engineering give insufficient support for refining high-level requirements to a concrete and assessable level. Furthermore, reuse mechanisms for these detailed requirements are missing. This paper proposes a web security model based on experiences with other quality models that is used in a security requirements engineering approach. The model provides (1) a means for refinement and (2) a requirements repository for reuse. The approach is illustrated with an example involving the Tomcat servlet container.

1 Introduction

Malicious attacks on software systems are a topic with high public visibility as average users can be affected. The level of vulnerabilities is still high today. The CERT[1] reports of 6,058 total vulnerabilities for the first 9 months of 2008. These attacks have a strong financial impact. In last year's E-Crime Watch Survey[2], it is stated that on average for each company security attacks result in a monetary loss of $456,700 in 12 months.

Therefore, software security is still a large and important problem in practice. It not only affects financial aspects but also ethical issues such as privacy. Hence, it is an important goal in software development to produce *secure* systems. This especially holds for web systems that are usually accessible in public networks. Secure web systems begin with the specification of security requirements. In order to become effective, they need to be clear and precise so that they are a real guidance for their implementation. This way, vulnerabilities can be prevented or at least reduced.

* This work has been supported in part by the German Federal Ministry of Education and Research (BMBF) in the project QuaMoCo (01 IS 08023B) and the German Academic Exchange Service (DAAD).

[1] http://www.cert.org/stats/vulnerability_remediation.html

[2] http://www.csoonline.com/documents/pdfs/e-crime_release_091107.pdf

Problem Despite the importance of high-quality security requirements for web systems, in practice they are often not well documented or not documented at all. Quality in general and security in particular are concepts with many facets and aspects. Hence, formulating precise requirements is difficult and elaborate.

First, there is no methodological guidance for refining high-level security requirements such as confidentiality or integrity to a concrete and assessable level. The refinement of security requirements is often complicated by many reasons such as by unavailable end users. This especially is true for systems that are distributed within a huge market, as it is the case for many web systems. Second, it is often unclear how reuse of requirements can be properly integrated in the development process. It is economically questionable to write detailed security requirements for each system from scratch.

Contribution The main contribution of this paper is an approach for security requirements engineering for web systems that mitigates the above mentioned problems. It uses an explicit and detailed security model as a knowledge repository for the reuse of requirements and their refinement. This security model builds on experiences with other quality models and documents in detail what security means in a given setting. We use the model in the approach to support two steps in particular: (1) deriving misuse cases based on the modelled attack patterns and (2) refinement of the high-level misuse cases to assessable, low-level requirements

Related Work In recent years much work has been done considering security requirements and related engineering processes. SQUARE [1] and SREP [2] describe activities to elicit and analyse security requirements. Misuse case driven approaches also contribute to the security requirements process [3]. Our approach builds on that and adds an activity-based security model as means for refinement and reuse. Although there exist many approaches for RE that are specifically elaborated for web systems [4], approaches that address the elicitation and refinement of security requirements are still missing. A similar situation exists with quality models. There are several interesting approaches involving web quality models, e.g. [5, 6]. However, none of these considers security explicitly.

2 Web Security Model

Quality models describe in a structured way what quality of software means. We introduce activity-based quality models and propose the web security instance that we use in the requirements approach.

2.1 Activity-Based Quality Models

We use the term *quality model* here in the sense of a *quality definition model* from [7], i.e. quality models define what quality is for a software system. However, in

practice this is often reduced to single metrics such as *number of defects* or high-level descriptions as given by the ISO 9126 [8]. These existing quality models have broadly acknowledged problems [9].

We have proposed to use activity-based quality models (ABQM) [10, 11] in order to address the shortcomings of existing quality models. The idea is to avoid using high-level "-ilities" for defining quality but to break it down into detailed facts and their influence on activities performed on and with the system. In addition to information about the characteristics of a system, the model contains further important facts about the process, the team and the environment and their respective influence on activities.

For ABQMs, an explicit meta-model was defined in order to characterise the quality model elements and their relationships. Four elements of the meta-model are most important: Entity, ATTRIBUTE, Impact and Activity. An entity can be any thing, animate or inanimate, that can have an influence on the software's quality, e.g. the source code of a method or the involved testers. These entities are characterised by attributes such as STRUCTUREDNESS or CONFORMITY. The combination of an entity and an attribute is called a *fact*. We use the notation [Entity | ATTRIBUTE] for a fact. These facts are assessable either by automatic measurement or by manual review. If possible, we define applicable metrics for measuring the facts inside the ABQM. An influence of a fact is then specified by an Impact. The impact on an Activity can be positive or negative. An activity is anything that is done with the system. For example, redundant methods (code clones) in the source code render modifications more difficult and elaborate. This is expressed in the model as [Method | REDUNDANCY] $\xrightarrow{-}$ [Modification].

The model does not only contain these impacts of facts on activities but also the relationships among these. Facts as well as activities are organised in hierarchies. A top-level activity Activity has sub-activities such as Use, Maintenance or Administration. In realistic quality models, these are then further refined.

Having defined all these entries in the ABQM, we can specify which activities we want to support and which influencing facts need to be considered. In terms of the above example, if we want to support the activity Modification, we know that we need to inspect the redundancy of methods.

2.2 The Web Security Instance

For handling web security requirements, we need to create a web security instance of the ABQM. Most important in this instance is to add attacks to the activity tree. These are activities that need to be negatively influenced. First, we have to differentiate between anticipated attacks and unanticipated attacks. A major problem in software security is that it is impossible to know all attacks that the system will be exposed to. This is the case because new attacks are developed every day. Hence, to assure quality, we need to employ two strategies: (1) prepare the system against anticipated attacks and (2) harden the system in general to avoid other vulnerabilities. For the classification of the attacks, there are several sources possible. We rely on the open community effort *Common Attack Pattern Enumeration and Classification* (CAPEC) [12] that is led

by the U.S. Department of Homeland Security. In the CAPEC, existing attack patterns are collected and classified. The attacks are organised in a hierarchy that is adapted in the activities tree (see Fig. 1).

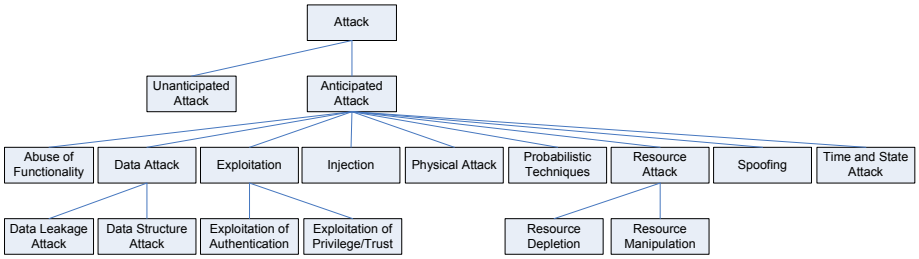

Fig. 1. The upper layers of the attack sub-tree of the activity tree

The creation of the facts tree is far more complicated. The facts tree needs to contain the available knowledge about characteristics of the system, its environment and the organisation that influence the described attacks. We employed various sources for collecting this knowledge including the ISO/IEC 27001 [13], the web guidelines[3], OWASP [14], and the Sun Secure Coding Guidelines for the Java Programming Language [15]. However, two main sources were used because they constitute large community efforts and hence provide consolidated knowledge: specific parts of the *Common Criteria* (CC) [16] and the *Common Weakness Enumeration* (CWE) [17]. The Common Criteria describe requirements on a system that should ensure security with a focus on what the system "shall do". The CWE looks at security from the other direction and describes reoccurring weaknesses in software systems that lead to vulnerabilities that are exploited by attacks. Therefore, these two sources combined give a strong basis for the facts tree.

We cannot describe the incorporation of the sources in all details in this paper but we give some examples on how knowledge from the sources has been modelled in our security model. For this, we use a sub-tree of the fact tree for the system as depicted in Fig. 2. The system consists of Data and Functionality. Furthermore, it has Dynamics and a Static Structure. These entities have then again children. For example, data can be a Cookie or an HTTP Request. Interesting functionality can be Cryptographic Support or File Handling.

Many of the entries in the quality model that have their origin in the Common Criteria are modelled as a part of Functionality because they mainly describe behavioural aspects that nevertheless are important for security. An example that is shown in Fig. 2 is the cryptographic support of the system. Following the CC, this can be decomposed into Cryptographic Key Management and Cryptographic Operation. A further part of Cryptographic Key Management is the Cryptographic Key Generation. The CC defines a requirement for that key generation that it shall be

[3] `http://www.webguidelines.nl/`

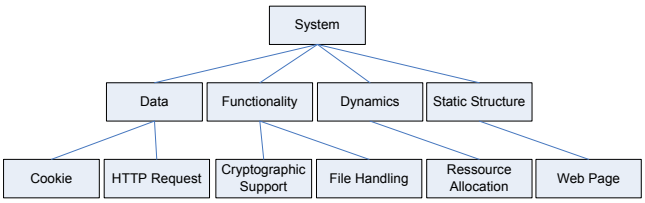

Fig. 2. Example entries of the system sub-tree from the fact tree

in accordance with a specified algorithm and specified key sizes. In the model, we express that by using the attribute APPROPRIATENESS for Cryptographic Key Generation. The resulting fact [Cryptographic Key Generation | APPROPRIATENESS] is textually described by "The system generates cryptographic keys in accordance with a specified cryptographic key generation algorithm and specified cryptographic key sizes that meet a specified list of standards." Unfortunately, the CC does not contain any description of impacts. This would make the standard more useful because the motivation to use these requirements would be higher. Hence, we complete the information using other sources. In this case, the CAPEC contains possible solutions and mitigations in the description of the *cryptanalysis* attack that includes the recommendation to use proven cryptographic algorithms with recommended key sizes. Therefore, we include the corresponding negative impact of [Cryptographic Key Generation | APPROPRIATENESS] on Cryptanalysis.

In contrast to the CC, the *Common Weakness Enumeration* mainly provides characteristics of the system and especially the kind of code that should be avoided. We include these characteristics into the model in this negative way with a positive influence on the attacks, i.e. making attacks easier. Another possibility is to reformulate the weaknesses as strength that are needed with negative influence on attacks. We used both possibilities depending on which option was more straightforward to model.

Several weaknesses in the CWE are not aiming at specific attacks but describe characteristics that are indicators for possible vulnerabilities. We model these as facts that have an impact on unanticipated attacks. An example from the CWE that is in our security model is *dead code*. Several parts of the code can be superfluous such as variables, methods or complete classes. For a variable, we can model that as a positive impact of [Variable | SUPERFLUOUSNESS] on Unanticipated Attack.

3 Security Requirements Approach

A requirements engineering (RE) process in general aims at systematically and effectively defining requirements that are aligned with the needs of all relevant stakeholders. According to [18] a RE process consists of the activities *elicitation*, *analysis* (refining requirements over several stages) and finally *validation*. What specific fine-grained techniques and approaches are used for each of these activities strongly depends on the application domain. We proposed a requirements

approach that makes use of the ABQM to reuse and refine requirements in [19]. Figure 3 illustrates the instance of the approach for the application domain of security requirements for web systems. The figure gives an overview of the activities, while the tasks within each of the activities are chosen according to the least common denominator of known security-specific RE approaches. We refer in particular to the Security Requirements Engineering Process (SREP) [2], the Security Quality Requirements Engineering Process (SQUARE) [1] and derivations of these approaches that integrate misuse cases [3].

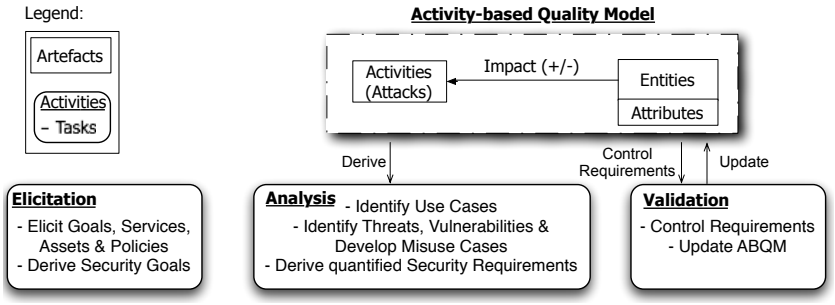

Fig. 3. Integration of the activity-based security model into an RE process

Elicitation During the requirements elicitation high level business requirements and/or market needs are collected. These represent high-level, initially stated, requirements. In particular, business requirements encompass (1) goals that have to be achieved by the final product, (2) services and assets that are offered and maintained by the product and (3) policies that might restrict its functionality. This information provides a sufficient basis for deriving security goals.

Analysis The requirements analysis aims at refining the business requirements to measurable ones. The first step of this refinement procedure consists of the identification of use cases. They represent the system's external behaviour by describing scenarios — specific sequences of interaction between users and the system. Because for security requirements also undesired use is relevant, misuse cases are derived. A misuse case can be seen as the inverse of a use case that shows an undesirable sequence of interactions, i.e. they describe how a specific attack can be performed.

The last step within analysis consists in deriving more concrete security requirements. These requirements demand specific properties (attributes) of the system and its environment that are meant to prevent the attack possibilities described within the misuse cases. For example, if a misuse case describes a specific attack that embeds malicious code into the system, derived security requirements could demand specific properties of sensitive data that is transmitted in encrypted connections.

However, as already described, a major challenge consists not only in identifying possible misuse cases but also in deriving security requirements from the

scenarios that address the prevention of such attacks. The quality model therefore serves as a knowledge repository that supports the reuse of requirements for the purpose of identifying and refining them. The activities within the quality model correspond to the scenarios (use cases and misuse cases) that are elicited and refined. The entities correspond to the system elements that are constrained by the requirements in terms of demanding specific properties of these elements, while the attributes of the entities correspond to such properties.

In this sense, the ABQM can be used to identify misuse cases by harvesting relevant attack scenarios from the activity tree. To derive quantified security requirements from these misuse cases, the ABQM is also used. As it also defines the relations between the scenarios and corresponding (e.g. technological) entities of a system, we only need to follow the impact of an activity to the corresponding entities to derive detailed requirements. These detailed requirements constrain the system's entities in a measurable way. In this case measurable does not necessarily mean automatically measurable. Also manual assessments like reviews and inspections can be used to evaluate quality requirements. This tackles the problem that requirements engineers do not need to have all technological possibilities in mind for each scenario. For example, attack scenarios that exclusively refer to web technologies can be grouped together [2]. As a whole, the analysis process based on the ABQM supports sufficient completeness of the quality requirements as this repository can be taken as the backbone of requirements analysis.

Validation The validation aims at proving and controlling requirements according to chosen (quality) criteria. In particular, the goal is to avoid and resolve conflicted, under-specified, unfeasible, incomplete and incorrect requirements. To find and correct such requirements, the ABQM can be used. For example, two requirements are conflicted (make contrary assertions), if they constrain the same system entity in a different way. Also under-specified requirements can be detected. If a requirement is under-specified, it is impossible to find an impact to a certain system entity. Also the correctness of the requirements can be tackled, because all detailed requirements are derived from business requirements and can be traced back to them.

Finally, requirements validation also includes updating the ABQM with requirements that are elaborated in addition to ones that are already stored within the ABQM. This update consists of two steps. First, the activity tree undergoes an update by inserting new scenarios, such as misuse cases that are not yet addressed by the activity tree. Second, the entities are inserted by adding the entities that are constrained with specific quality attributes.

4 Example

To further illustrate our approach, we use the servlet container *Tomcat 6.0*[4] and develop security requirements for it. Tomcat is the reference implementation of

[4] `http://tomcat.apache.org/`

the Java servlet and JSP specification and provides support to deliver dynamically assembled HTTP responses based on an HTTP request. We follow the RE process introduced in the previous section, while describing only exemplary requirements for reasons of space limitations.

Elicitation We concentrate on the most important high level *goal* for Tomat: The system allows the user to send any request and get a response. From this goal the elementary services *request* and *response* are derived in terms of that a server has to respond to every possible sequences of HTTP requests from a client. Several critical assets are possible but we assume that only *sensitive information* is important in our example. Security goals are now identified to protect each of the identified services and assets and to attain the defined goals. The derived security goals are: (a) The system services shall be available for the subscript user and (b) the system shall ensure privacy, integrity and availability of the sensitive data communicating among the involved legitimate parties.

Analysis Based on the elicited business requirements we examine the threats and vulnerabilities from our model. We use the identified services as a basis for identifying the relevant use case (request & response). There are several possible anticipated attacks for this use case that violate the confidentiality and integrity of sensitive information. For instance, injection attacks that cause the unauthorised access in an established session so that attackers can view, capture and modify any data that is communicated within this session. Therefore, "attack by injection" is considered as a misuse case. Figure 4 illustrates how from the ABQM misuse cases are derived. The left side of the figure represents the attack activities contained in the model, the right side the identified misuse cases that set the attacks in context to the relevant use cases. In the model, this attack is influenced by the fact [Directive | SANITATION] and [Processing | SANITATION]. Hence, the requirements have to explicitly include that user input of directives and processing of data needs to be validated before it is used. In a similar way, we analyse the other attacks such as resource attacks with their related facts.

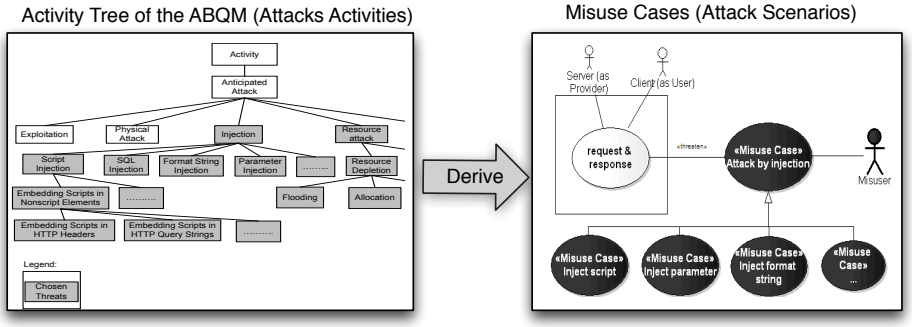

Fig. 4. Using the ABQM for deriving misuse cases

At this stage security requirements are specified based on the identified attacks and related facts from the model. These security requirements are detailed enough to be able to assess their fulfilment directly in the code, e.g. by reviews. The elicited security requirements can be textually specified like in the following:

- The software shall sanitise user-controllable input for content before it is prepared in output that is used as a web page. Unsanitized special elements that have control implications in web pages, such as HTML tags or mouse events, are interpreted as control characters that execute in violation of the client's trust in the application or system. This weakness usually enables cross-site scripting attacks in web applications.
- The system shall verify user inputs that are assumed to be immutable but are actually externally controllable, such as a cookie.

Validation As our short list of the above requirements does not contain any conflicts, the validation is rather simple. However, there are further threats which are not currently covered in our activity tree. For example, an attacker can have the opportunity to view or modify the sensitive data by unauthorised physical means. This needs to be refined without specific guidance from the ABQM but can then be fed back into it. This way, this new kind of attack is available in the requirements elicitation for the next project.

5 Conclusions

Because of the importance of software security in web systems, we need to consider security aspects from the very beginning in the requirements engineering process. The basis of our security requirements approach consists of an activity-based security model that uses experiences with such models in other areas. The way of modelling allows to break down security to assessable and partly directly measurable characteristics of the system, its environment and the organisation. The activities in the model consist of common attack patterns that need to be prevented and thereby deliver the means for refining high-level security goals to such concrete and assessable, but necessarily measurable characteristics. Moreover, the model can also serve as a comprehensive repository that fosters the reuse of security requirements.

An example was conducted with the Tomcat servlet container. It showed that the approach is feasible in principle and that there is a potential for preventing vulnerabilities by specifying more concrete requirements. We plan to perform industrial case studies with the security model in order to explore the benefits and limitations of the approach. Finally, the UMD by Donzelli and Basili [20] will be analysed for a potential extension of our approach because it contains some additional aspects that might be useful for requirements elicitation.

References

1. Mead, N., Steheny, T.: Security quality requirement engineering methodology. In: Proc. Workshop on Software Engineering for Secure Systems (SESS '05). (2005)

2. Mellado, D., Medina, E., Piattini, M.: Acommon criteria based security requirements engineering process for the development of secure information system. Computer standards & interfaces **29** (June 2007) 244–253
3. Sindre, G., Opdahl, A.: Eliciting security requirements with misuse case. Requirements Engineering Journal **44**(10) (June 2005) 34–44
4. Koch, N., Escalona, J.: Requirements engineering for web applications - a comparative study. Journal of Web Engineering **2**(3) (2004)
5. Ruiz, J., Calero, C., Piattini, M.: A Three Dimensional Web Quality Model. Volume 2722 of LNCS. Springer-Verlag (2003)
6. Malak, G., Badri, L., Badri, M., Sahraoui, H.: Towards a Multidimensional Model for Web-Based Applications Quality Assessment. Volume 3182 of LNCS. Springer-Verlag (2004)
7. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: Proc. 7th International Workshop on Software Quality (WoSQ '09), IEEE Computer Society (2009)
8. : ISO 9126: Product Quality – Part 1: Quality Model (2003)
9. Kitchenham, B., Pfleeger, S.L.: Software quality: The elusive target. IEEE Softw. **13**(1) (1996) 12–21
10. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proc. 23rd International Conference on Software Maintenance (ICSM '07), IEEE Computer Society (2007) 184–193
11. Winter, S., Wagner, S., Deissenboeck, F.: A comprehensive model of usability. In: Proc. Engineering Interactive Systems 2007 (EIS '07). Volume 4940 of LNCS., Springer (2008)
12. Homeland Security: Common attack pattern enumeration and classification (CAPEC). Available Online at http://capec.mitre.org/. Accessed in October 2008
13. : ISO/IEC 27001: Information technology – Security techniques – Information security management systems – Requirements (2005)
14. Wiesmann, A., van der Stock, A., Curphey, M., Stirbei, R., eds.: A Guide to Building Secure Web Applications and Web Services. OWASP (2005)
15. Sun Microsystems: Secure coding guidelines for the java programming language, version 2.0. Available Online at `http://java.sun.com/security/seccodeguide.html`
16. : Common criteria for information technology security evaluation, version 3.1. Available Online at `http://www.commoncriteriaportal.org/`
17. Homeland Security: Common weakness enumeration (CWE). Available Online at http://cwe.mitre.org/. Accessed in October 2008
18. Wiegers, K.E.: Software Requirements. Microsoft Press, Redmond, WA, USA (2003)
19. Wagner, S., Deissenboeck, F., Winter, S.: Managing quality requirements using activity-based quality models. In: WoSQ '08: Proceedings of the 6th international workshop on Software quality, New York, NY, USA, ACM (2008) 29–34
20. Donzelli, P., Basili, V.: A practical framework for eliciting and modeling system dependability requirements: Experience from the NASA high dependability computing project. The Journal of Systems and Software **79** (2006) 107–119