Assessing the Usefulness of Testing for Validating and Correcting Security Risk Models Based on Two Industrial Case Studies

Gencer Erdogan, SINTEF ICT and University of Oslo, Norway

Fredrik Seehusen, SINTEF ICT, Norway

Ketil Stølen, SINTEF ICT and University of Oslo, Norway

Jon Hofstad, EVRY, Norway

Jan Øyvind Aagedal, Accurate Equity, Norway

ABSTRACT

We present the results of an evaluation in which the objective was to assess how useful testing is for validating and correcting security risk models. The evaluation is based on two industrial case studies. In the first case study we analyzed a multilingual financial Web application, while in the second case study we analyzed a mobile financial application. In both case studies, the testing yielded new information which was not found in the risk assessment phase. In particular, in the first case study, new vulnerabilities were found which resulted in an update of the likelihood values of threat scenarios and risks in the risk model. New vulnerabilities were also identified and added to the risk model in the second case study. These updates led to more accurate risk models, which indicate that the testing was indeed useful for validating and correcting the risk models.

Keywords: Security Risk Analysis, Security Testing, Test-driven Security Risk Analysis, Case Study

INTRODUCTION

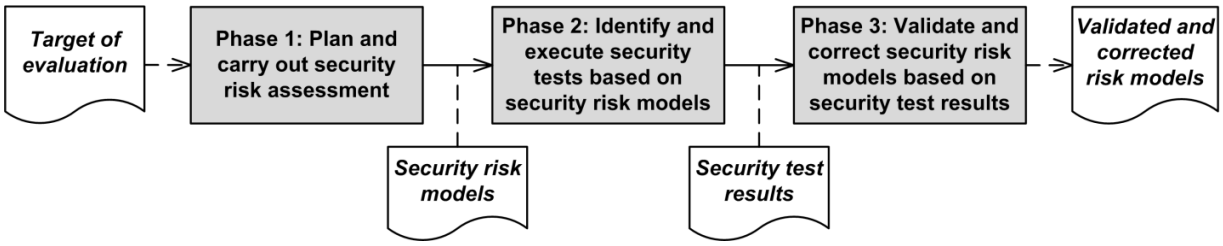
Security risk analysis is carried out in order to identify and assess security specific risks. Traditional risk analyses often rely on expert judgment for the identification of risks, their causes, as well as risk estimation in terms of likelihood and consequence. The outcome of these kinds of risk analyses is therefore dependent on the background, experience, and knowledge of the participants, which in turn reflects uncertainty regarding the validity of the results.

In order to mitigate this uncertainty, security risk analysis can be complemented by other ways of gathering information of relevance. One such approach is to combine security risk analysis with security testing, in which the testing is used to validate and correct the risk analysis results. We refer to this as test-driven security risk analysis.

We have developed an approach to test-driven security risk analysis, and as depicted in Figure 1, our approach is divided into three phases. Phase 1 expects a description of the target of evaluation. Then, based on this description, the security risk assessment is planned and carried

out. The output of Phase 1 is security risk models, which is used as input to Phase 2. In Phase 2, security tests are identified based on the risk models and executed. The output of Phase 2 is security test results, which is used as input to the third and final phase. In the third phase, the risk models are validated and corrected with respect to the security test results.

Figure 1. Overview of the test-driven security risk analysis approach.



In this paper, we present an evaluation of our test-driven security risk analysis approach based on two industrial case studies. The objective of the case studies was to assess how useful testing is for validating and correcting security risk models. The basis of our evaluation is to compare the risk models produced before and after testing. That is, we compare the difference in risk models produced in Phase 1 with the updated risk models produced in Phase 3.

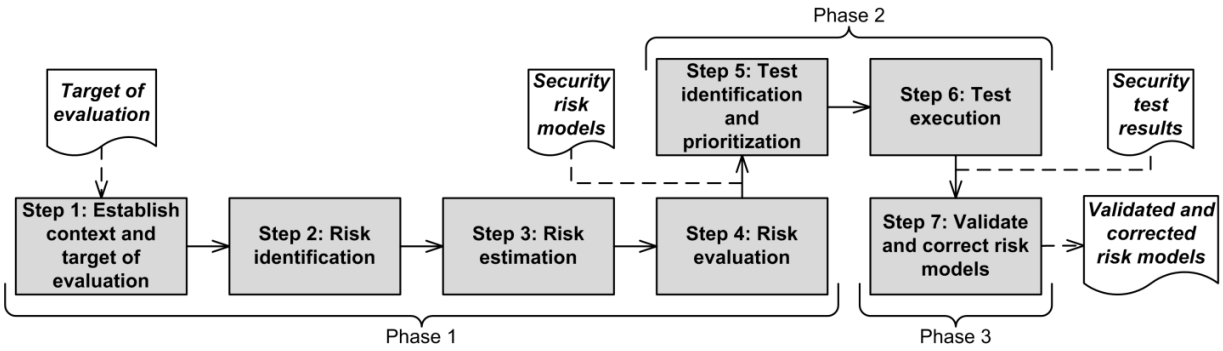
The first case study was carried out between March 2011 and July 2011, while the second case study was carried out between June 2012 and January 2013. In the first case study we analyzed a multilingual financial Web application, and in the second case study we analyzed a mobile financial application. The systems analyzed in both case studies serve as the backbone for the system owner's business goals and are used by a large number of users every day. The system owners, which are also the customers that commissioned the case studies, required full confidentiality. The results that are presented in this paper are therefore limited to the experiences from applying the test-driven security risk analysis approach.

The reminder of the paper is structured as follows. Section 2 describes our test-driven security risk analysis approach. Section 3 describes our research method. Section 4 gives an overview of the two case studies. Section 5 describes the results obtained in the case studies which are the basis of our evaluation. Section 6 provides a discussion of the results with respect to our research questions and overall hypothesis. Section 7 discusses related work. Finally, Section 8 highlights our key findings and concludes the paper.

TEST-DRIVEN SECURITY RISK ANALYSIS

As already illustrated in Figure 1, the test-driven security risk analysis process is divided into three phases. Each phase is further divided into various steps. Figure 2 shows the steps within each phase. In the following, we give a more detailed description of each phase.

Figure 2. The steps in test-driven security risk analysis.



Security Risk Assessment (Phase 1)

The security risk assessment phase consists of four steps corresponding to the steps of the CORAS method, which is a model-driven approach to risk analysis (Lund, Solhaug, & Stølen, 2011). The purpose of the first step is to prepare the security risk assessment. The first step is carried out together with the customer and involves defining a precise scope and focus of the target of evaluation, defining security assets that needs to be taken into consideration during the risk assessment, defining likelihood and consequence scales necessary for risk estimation, as well as defining risk evaluation matrices based on the likelihood and consequence scales for evaluating risks.

Table 1 shows an example of a likelihood scale. In our approach, the term likelihood is a general description of the frequency for incidents to occur, and the likelihood scale defines the values that will be used when assigning likelihood estimates to unwanted incidents in the risk model. Because incidents may have different impact depending on which security asset is harmed, a separate consequence scale is defined for each security asset taken into consideration. Let us assume that we have defined a security asset named *availability of service*. Table 2 shows an example consequence scale defined for this particular security asset, in which the consequence values are defined in terms of downtime. Another security asset could for example be *confidentiality of user data*. This security asset would have a different consequence scale concerning the disclosure of confidential user data such as login credentials, credit card information, and privacy-related data.

Table 1. Likelihood scale.

Likelihood value	Description	Definition
Rare	Less than once per year	$[0, 1\rangle$: 1y
Unlikely	One to five times per year	$[1, 5\rangle$: 1y
Possible	Five to twenty times per year	$[5, 20\rangle$: 1y
Likely	Twenty to fifty times per year	$[20, 50\rangle$: 1y
Certain	Fifty times or more per year	$[50, \infty\rangle$: 1y

Table 2. Consequence scale for asset: Availability of service.

Consequence value	Description
Insignificant	Downtime in range [0, 1 minute⟩
Minor	Downtime in range [1 minute, 1 hour⟩
Moderate	Downtime in range [1 hour, 1 day⟩
Major	Downtime in range [1 day, 1 week⟩
Catastrophic	Downtime in range [1 week, ∞⟩

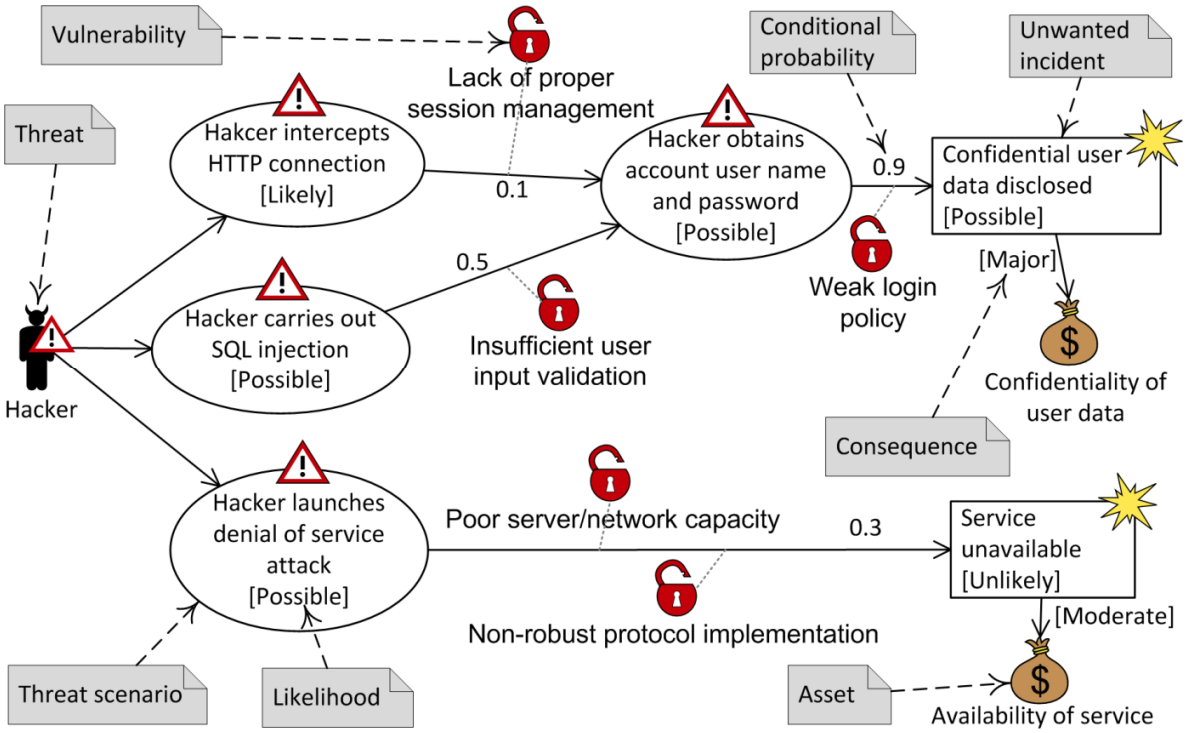
Having made the necessary preparations, we start carrying out security risk assessment. That is, risk identification, risk estimation, and risk evaluation. The risk models are constructed on-the-fly during the risk assessment process which is carried out in a series of workshops. The information obtained in these workshops is usually based on expert judgment, mostly from representatives of the customer on whose behalf the analysis is conducted.

During risk identification (Step 2), risks are identified by analyzing the target of evaluation and identifying potential unwanted incidents that may harm certain security assets. In addition, we identify threat scenarios causing the unwanted incidents, as well as threats initiating the threat scenarios. We carry out this process by modeling risk models using the CORAS modeling language. The constructed risk models are referred to as CORAS risk models. Figure 3 shows an example of a CORAS risk model, and as illustrated, a CORAS risk model is a directed acyclic graph where every node is of one of the following kinds.

- **Threat**: A potential cause of an unwanted incident.
- **Threat scenario**: A chain or series of events that is initiated by a threat and that may lead to an unwanted incident.
- **Unwanted incident**: An event that harms or reduces the value of an asset.
- **Asset**: Something to which a party assigns value and hence for which the party requires protection.

Risks can also be represented in a CORAS risk model. These correspond to pairs of unwanted incidents and assets. If an unwanted incident harms exactly one asset, as illustrated in Figure 3, then the unwanted incident will represent a single risk. If an unwanted incident harms two assets, then the unwanted incident represents two risks, etc.

Figure 3. Example of a CORAS risk model.



As part of risk identification, we also identify vulnerabilities that may be exploited by a threat in order to cause security risk. Vulnerabilities are modeled as open locks, and are attached on relations in a CORAS risk model. A relation in a CORAS risk model may be one of the following kinds.

- **Initiates relation**: A relation that goes from a threat *A* to a threat scenario or an unwanted incident *B*, meaning that *A* initiates *B*.
- **Leads to relation**: A relation that goes from a threat scenario or an unwanted incident *A* to a threat scenario or an unwanted incident *B*, meaning that *A* leads to *B*.
- **Harms relation**: A relation that goes from an unwanted incident *A* to an asset *B*, meaning that *A* harms *B*.

Vulnerabilities may be assigned on the initiates relations or the leads-to relations going from *A* to *B*, describing a weakness, flaw or deficiency that opens for *A* leading to *B*.

Having identified risks, the next step is to estimate the likelihood of risks occurring, as well as the consequence of risks (Step 3). This step makes use of the predefined likelihood and consequence scales (Table 1 and Table 2 in this example). The likelihood estimation is carried out by first estimating the likelihood of threat scenarios causing the risks, which also includes the estimation of conditional probabilities. Then, based on these estimates, we calculate the likelihood of unwanted incidents. During the calculation it is important to consider whether the threat scenarios are mutually exclusive or independent, and make sure that the calculations are carried out in a consistent manner. However, it is beyond the scope of this paper to explain the

rules for calculating likelihood values using CORAS risk models. The reader is referred to the CORAS approach (Lund et al., 2011) for a precise explanation of the calculation rules. Finally, to complete risk estimation, the consequence for each risk is estimated by making use of the predefined consequence scale. Thus, relations and nodes may have the following risk-measure annotations.

- **Likelihood values**: May be assigned to a threat scenario or an unwanted incident *A*, estimating the likelihood of *A* occurring.
- **Conditional probabilities**: May be assigned on the leads to relations going from *A* to *B*, estimating the probability that *B* occurs given that *A* has occurred.
- **Consequence values**: May be assigned on the harms relations going from *A* to *B*, estimating the consequence that the occurrence of *A* has on *B*.

Having estimated risks, the next step is to evaluate the risks with respect to their likelihood and consequence values in order to determine a *risk level* for each identified risk (Step 4). The risk level is calculated by plotting a risk into a *risk matrix* with respect to its likelihood and consequence value. Risk evaluation matrices are constructed with respect to predefined likelihood and consequence scales, and are typically divided into three risk levels: Low, medium, and high, which is commonly used as a basis to decide whether to accept, monitor, or treat risks, respectively. Table 3 shows an example of a risk evaluation matrix constructed with respect to the likelihood scale in Table 1, and the consequence scale in Table 2. That is, Table 3 shows a risk evaluation matrix constructed for evaluating risks that impact the security asset *availability of service*. As shown in Table 3, the cells of the matrix are divided into three groups in order to represent low, medium, and high risks. For example, by mapping the risk *service unavailable* modeled in Figure 3 to the risk evaluation matrix, we see that the risk is assigned a *low* risk level. Similarly, if we map the risk *confidential user data disclosed* to a similar risk matrix, then it is assigned a *high* risk level.

Table 3. Risk evaluation matrix.

		Consequence				
		Insignificant	**Minor**	**Moderate**	**Major**	**Catastrophic**
Likelihood	**Rare**	*low*				
	Unlikely			**Service unavailable**		
	Possible				**Confidential user data disclosed**	
	Likely					
	Certain	*medium*				*high*

In traditional risk analysis, the next step is to decide which risks to treat, based on the risk evaluation matrix, and then suggest treatments for those risks. The process of risk treatment is also carried out in our approach. However, before risks treatment, we test the obtained risk models in order to check their validity. This is explained in detail in the next two sections, but

the main idea is as follows. In case the test results do not match the information conveyed by the risk models, we correct the risk models according to the test results. In case the test results match the information conveyed by the risk models, we do not make any changes. Either way, the testing increases our confidence in the validity of the risk models, which in turn increases the confidence of the decisions made related to risk treatment.

The reader is referred to Lund et al. (2011) for further explanation on the steps of Phase 1. Throughout the paper, we use the term *risk model element*, or just *element* for short, to mean a node, a relation, or an assignment, and we will sometimes use the terms unwanted incident and risk interchangeably when the distinction is not important.

Security Testing (Phase 2)

The security testing phase consists of two steps, test identification and prioritization (Step 5), and test execution (Step 6). In Step 5, test cases are designed by first identifying high-level test procedures based on CORAS risk models. Then, the test procedures are prioritized and manually refined into concrete test cases.

A CORAS risk model can be seen as a set of statements about the world. One way of identifying test procedures based on a CORAS risk model is to derive statements with respect to the threat scenarios in the risk model. CORAS provides algorithms that may be used to translate a risk model into English prose (Lund et al., 2011), and we use these algorithms in our approach to construct test procedures. Typically, a threat scenario corresponds to a particular type of attack, and the purpose of testing a threat scenario is to find vulnerabilities that may be exploited by the attack. Based on the risk model in Figure 3 we may identify the following test procedures.

- Check that *Hacker* initiates *Hacker intercepts HTTP connection* with likelihood *Likely*.
- Check that *Hacker* initiates *Hacker carries out SQL injection* with likelihood *Possible*.
- Check that *Hacker* initiates *Hacker launches denial of service attack* with likelihood *Possible*.
- Check that *Hacker intercepts HTTP connection* leads to *Hacker obtains account user name and password* with conditional probability *0.1*, due to vulnerability *Lack of proper session management*.
- Check that *Hacker carries out SQL injection* leads to *Hacker obtains account user name and password* with conditional probability *0.5*, due to vulnerability *Insufficient user input validation*.
- Check that *Hacker launches denial of service attack* leads to *Service unavailable* with conditional probability *0.3*, due to vulnerabilities *Poor server/network capacity* and *Non-robust protocol implementation*.
- Check that *Hacker obtains account user name and password* leads to *Confidential user data disclosed* with conditional probability *0.9*, due to vulnerability *Weak login policy*.

Having identified test procedures as described above, we prioritize them and select those that are considered most important to test (that is, we do not test all the threat scenarios of the risk

models). The priority of a threat scenario, and thereby the priority of its corresponding test procedure is assessed by estimating its severity, as well as the required effort to implement and carry out the test procedures in terms of time.

Severity is an estimate of the impact that a threat scenario has on the identified risks. The intuition is that a high degree of impact should result in a high priority. In the extreme case, if the threat scenario has zero impact on a risk, then there is no point in testing it. The severity is also affected by our confidence in the correctness of the threat scenario. Intuitively, the less confident we are about the correctness of the threat scenario, the more it makes sense to test it. If we, however, are completely confident in the correctness of the threat scenario, then there is no point in testing it because then we strongly believe that this will not give us any new information.

The severity estimation is carried out as follows. First, we express the conditional probabilities as conditional probability intervals in order to reflect our confidence in the correctness of the threat scenarios. For example, considering the example in Figure 3, threat scenario *Hacker carries out SQL injection* leads to threat scenario *Hacker obtains account user name and password* with conditional probability *0.5*. If we are completely certain that the conditional ratio is *0.5*, then we express this as an interval in terms of *[0.5, 0.5]* representing the presumed minimum and maximum probability, respectively. As mentioned above, in this case there is no point in testing the threat scenario because we are certain about its correctness. However, if we are uncertain about the correctness of the threat scenario and would like to express an interval considering probabilities between *0.4* and *0.6*, we write *[0.4, 0.6]*. The width of the interval expresses the uncertainty. That is, the wider the width between the minimum probability and the maximum probability, the less confident we are about the estimate. Second, for each probability interval, we recalculate the severity of the threat scenario in question with respect to the minimum probability, and then with respect to the maximum probability. In the former we obtain minimum severity for the threat scenario in question, while in the latter we obtain the maximum severity. Third, we subtract the maximum severity from the minimum severity and obtain a value which is used as the priority of the threat scenario in question. We do this for all threat scenarios in the risk model and obtain a prioritized list of threat scenarios, that is, a prioritized list of test procedures.

Having prioritized the test procedures using the severity score, we select those procedures that will be implemented by also taking effort into account. Effort is an estimate of the time it will take to implement and carry out the corresponding test procedure. It is important to consider effort, because in some situations the effort required to carry out certain test procedures may be outside the boundaries of available time and budget. In such cases, the customer has to make a cost-benefit analysis and decide whether or not to carry out the test procedures. Moreover, it is often very difficult to test whether a threat actually will carry out an attack in the first place; at least this is difficult to test using conventional testing techniques. The first three test procedures listed above are examples of such test procedures, and in this case we would exclude them from the test selection.

The priority of a test procedure is carried out as described above. It is beyond the scope of this paper to give a detailed explanation of this process, and we therefore refer the reader to our earlier work for a more detailed explanation (Seehusen, 2014). After selecting the test procedures

with the highest priority, they are refined into concrete test cases by making a detailed description of how they should be tested. Step 6, test execution, is the activity of executing the tests identified in Step 5, as well as recording the test results. The test cases are typically executed on a test environment provided by the customer.

Validation and Correction of Risk Models (Phase 3)

The validation and correction phase consists of one step (Step 7), which has two main purposes. The first is to validate and correct the risk models (output of Phase 1) with respect to the test results (output of Phase 2). The second is to identify treatments for the updated risk models. This step is based on the corresponding step of the CORAS method. However, most of the reminder of this paper will be devoted to explaining the degree to which we had to update the risk models by taking the test results into consideration.

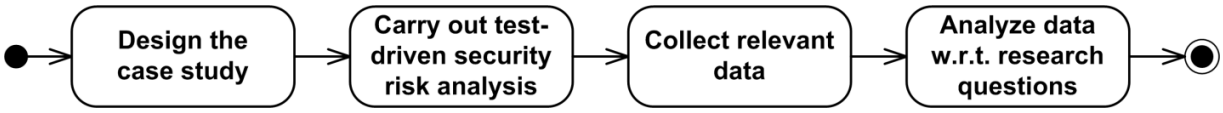
Based on the test results, we may validate the risk models by checking whether the test results correspond to the information conveyed by the risk models. If the test results do not correspond to the information conveyed by the risk models, then we may correct the risk models with respect to the test results in terms of adding, deleting or editing risk model elements. Let RM_B be the risk model before testing, and RM_A be the risk model after testing. The following points describe what we mean by adding, deleting and editing risk model elements based on the test results.

- **Add**: An element in RM_A has been added if it is not in RM_B.
- **Delete**: An element in RM_B has been deleted if it is not in RM_A.
- **Edit**: A node or relation in both RM_B and RM_A has been edited if its assignment in RM_B or RM_A is different.

RESEARCH METHOD

We conducted the case study research in four main steps. First, we designed the case study, which includes defining the objective, the units of analysis, as well as the research questions. Second, we carried out the test-driven security risk analysis approach within an industrial setting. Third, we collected the relevant data produced in the test-driven security risk analysis. Fourth, we analyzed the collected data with respect to our research questions. Figure 4 gives an overview of the main activities of the research method. Our research method is inspired by the guidelines for case study research in software engineering provided by Runeson et al. (2012). This research approach was applied in both case studies.

Figure 4. The main activities of the research method.



As mentioned in Section 1, the objective of the case studies was to assess the usefulness of testing for the purpose of validating and correcting the risk models. Although we carried out the complete test-driven security risk analysis approach in both case studies, we specifically focused on the degree to which the test results yielded information that caused us to update the risk

models. That is, the units of analysis are the risk models produced in Phase 1, and the validated and corrected risk models produced in Phase 3. Thus, the data that were collected and analyzed in the case studies were mainly the risk models produced in Phase 1 and Phase 3.

Our hypothesis is that security testing is useful for validating and correcting the security risk models. As explained in Section 2.3, the test results may be used to validate the risk models. However, if the test results do not correspond to the information conveyed by the risk models, then we may use the test results to correct the risk models in terms of adding, deleting, and editing risk model elements. Thus we define the following research questions.

- **RQ1**: To what extent are the test results useful for correcting the risk model by adding elements?
- **RQ2**: To what extent are the test results useful for correcting the risk model by deleting elements?
- **RQ3**: To what extent are the test results useful for correcting the risk model by editing elements?

OVERVIEW OF THE TWO CASE STUDIES

Both case studies were carried out in an industrial setting, and the analyzed systems were already deployed in production and used by a large number of users every day. As indicated in Section 2, we carried out workshops together with the representatives of the customers as part of our approach. However, the time devoted to the workshops was limited and mainly used to gather the necessary information, as well as to communicate our findings. Most of the time in the case studies was devoted to carry out the steps before and after the workshops.

Case Study 1: Test-Driven Security Risk Analysis of a Financial Web Application

In the first case study, we analyzed a multilingual web application which is designed to deliver streamlined administration and reporting of all forms of equity-based compensation plans. The web application was deployed on the servers of a third party service provider, as well as maintained by the same service provider with respect to infrastructure. However, the web application was completely administrated by the client commissioning the case study for business purposes, such as customizing the web application for each customer, as well as patching and updating features of the web application. The focus of this case study was to analyze the system to identify security risks that may be introduced internally by the client when administrating the application, as well as security risks that may be introduced externally via features available to customers. In this case study, it was decided not to consider security risks related to infrastructure because this was a contractual responsibility of the service provider.

Table 4 gives an overview of the workshops in the first case study in terms of time spent in each step of the approach before, during, and after the workshops. The table reflects the size of the case study in terms of number of participants, as well as the invested time in carrying out the approach in an industrial setting. The column *step* shows the steps of test-driven security risk analysis that were carried out in the corresponding workshop. In the *participant* column, the denotations C:*n* and A:*m* represent *n* participants from the customer side and *m* participants from the risk analysis team. The column *duration during workshop* shows the time spent in each workshop. The columns *duration before workshop* and *duration after workshop* give the

approximate time spent before and after each workshop, respectively. Notice that there was no workshop addressing Step 6 (test execution). This is because test execution was entirely conducted between the fifth and the sixth workshop, which is why the duration after workshop 5 is approximately 100 hours. The total time spent in the first case study was approximately 293 hours.

Table 4. Overview of the workshops in the first case study.

Workshop	Date	Step	Participant	≈ Duration before workshop	Duration during workshop	≈ Duration after workshop
1	28.03.2011	Step 1	C:1, A:3	≈ 8 hours	2 hours	≈ 10 hours
2	12.04.2011	Step 1	C:3, A:4	≈ 8 hours	3 hours	≈ 10 hours
3	09.05.2011	Step 1	C:2, A:3	≈ 10 hours	3 hours	≈ 10 hours
4	20.05.2011	Step 2 and 3	C:1, A:2	≈ 15 hours	6 hours	≈ 45 hours
5	27.05.2011	Step 4 and 5	C:1, A:2	≈ 15 hours	6 hours	≈ 100 hours
6	07.07.2011	Step 7	P:2, A:2	≈ 10 hours	2 hours	≈ 30 hours

During the case study, we made use of the CORAS tool (CORAS, 2015) to model CORAS risk models as well as to identify test procedures. Based on the test procedures, we designed the test cases and executed them automatically, semi-automatically and manually using the following tools.

- **Automated testing**: IBM Rational Software Architect (IBM, 2015), Smartesting CertifyIt (Smartesting, 2015), Selenium (Selenium, 2015), Eclipse (Eclipse, 2015).
- **Semi-automated testing**: OWASP Zed Attack Proxy (Open Web Application Security Project, 2015).
- **Manual testing**: Wireshark (Wireshark, 2015).

All the tests were executed at the level of the HTTP protocol, that is, at the application level, and from a black-box perspective of the system that was analyzed. The automated process was as follows. First we specified a model of the system under test using IBM RSA together with the CetifyIt plugin, then we used CetifyIt to generate abstract Java tests, then we concretized these in Java using Eclipse, and finally we executed them using the Selenium plugin on the Firefox Web browser (Mozilla, 2015).

The semi-automated process was carried out using the OWASP Zed Attack Proxy tool to intercept the HTTP requests and responses. Then we manually altered the information in the requests and responses. We used Wireshark to analyze the communication between the client and the server at the IP level.

Case Study 2: Test-Driven Security Risk Analysis of a Mobile Financial Application

In the second case study, we analyzed a mobile application designed to provide various online financial services to the users on their mobile devices. In contrast to the first case study, this application was deployed on the local servers of the client commissioning the case study. The online financial services were accessible only via a dedicated mobile application installed on a mobile device. Moreover, all aspects related to maintenance and administration was the responsibility of the client. However, some few aspects related to content displayed to the user were directly handled by a third party. The main focus of this case study was to analyze the mobile application to identify security risks that may be introduced from an external point of view. That is, we identified security risks that may be introduced by the third party when administrating the few aspects of the mobile application, as well as security risks that may be introduced via features available to customers.

Table 5 gives an overview of the workshops in the second case study, as well as the approximate time spent before and after each workshop. Similar to the first case study, there was no workshop addressing Step 6 (test execution), which was conducted between the fifth and sixth workshop. Notice that in the first case study, the three first workshops were dedicated to Step 1 (establish context and target of evaluation), while in the second case study we only needed one workshop for the same step. This is because in the first case study, the context and target of evaluation was not entirely clear in the beginning and needed to be concretized over several iterations. In the second case study, the context and target of evaluation were somewhat concretized by the customer prior to the first workshop, and it was therefore sufficient with one workshop for Step 1. In addition, we got access to the system documentation prior to the first workshop, which helped us to get a good understanding of the target of evaluation before the workshop. The total time spent in the second case study was approximately 322 hours. Moreover, the fact that we spent approximately 293 hours in the first case study and 322 hours in the second case study, in which the analyzed systems were different in terms of architecture and complexity, shows that the approach is applicable in industrial settings within reasonable time. This also gives an indication of how much time and effort may be required by the participants involved in the test-driven security risk analysis process in an industrial setting.

Table 5. Overview of the workshops in the second case study.

Workshop	Date	Step	Participant	≈ Duration before workshop	Duration during workshop	≈ Duration after workshop
1	19.06.2012	Step 1	C:7, A:4	≈ 20 hours	5 hours	≈ 25 hours
2	23.08.2012	Step 2	C:5, A:3	≈ 8 hours	6 hours	≈ 35 hours
3	21.09.2012	Step 3	C:4, A:3	≈ 8 hours	6 hours	≈ 35 hours
4	01.10.2012	Step 4	C:3, A:3	≈ 8 hours	2.5 hours	≈ 25 hours
5	16.11.2012	Step 5	C:6, A:3	≈ 10 hours	3 hours	≈ 85 hours
6	24.01.2013	Step 7	C:4, A:3	≈ 8 hours	2 hours	≈ 30 hours

Risk modeling, test procedure generation, test case design and the execution of the test cases were carried out in a similar manner as in the first case study. In addition to the tools used in the

first case study, we made use of the tool Burp Suite Free Edition (Portswigger, 2015) for automatic analysis purposes, and Google Chrome (Google, 2015) for manual tests.

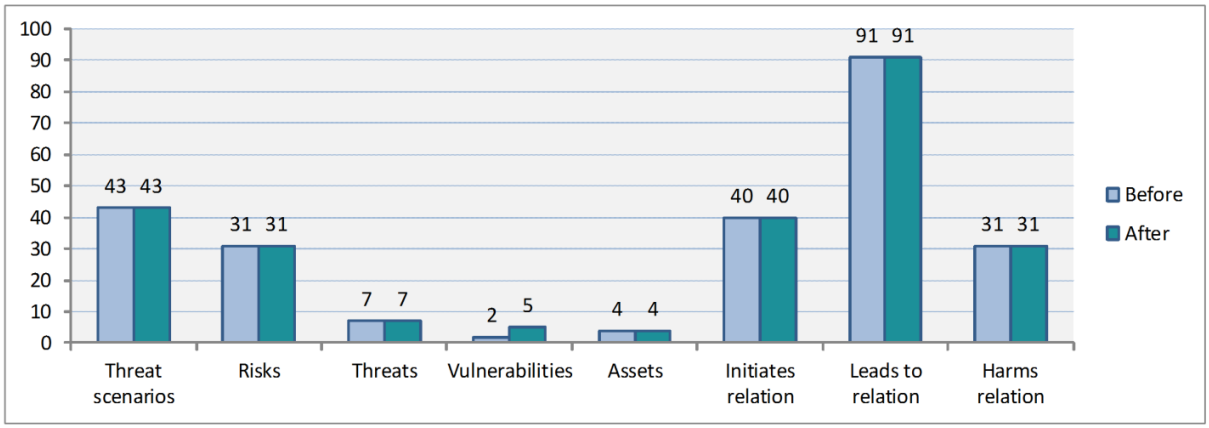
RESULTS

In this section, we present the results obtained in both case studies, and describe the difference between the risk models before and after testing for each case study.

Results Obtained in Case Study 1

After testing, no nodes or relations were added to or deleted from the risk model. The only risk model elements that were added and deleted were vulnerabilities (recall that vulnerabilities are not classified as nodes, but as assignments to relations). More precisely, one vulnerability was deleted and four vulnerabilities were added after testing. However, it is important to note that the deletion of vulnerabilities was not only based on the test results. In the case studies, the analyst removed vulnerabilities by taking the test results into consideration, as well as own experience and knowledge. That is, the vulnerabilities were removed from the risk models based on expert judgement, and not purely based on the test results. In Figure 5 we have illustrated the number of risk model nodes, the number of vulnerabilities, and the number of relations before and after testing.

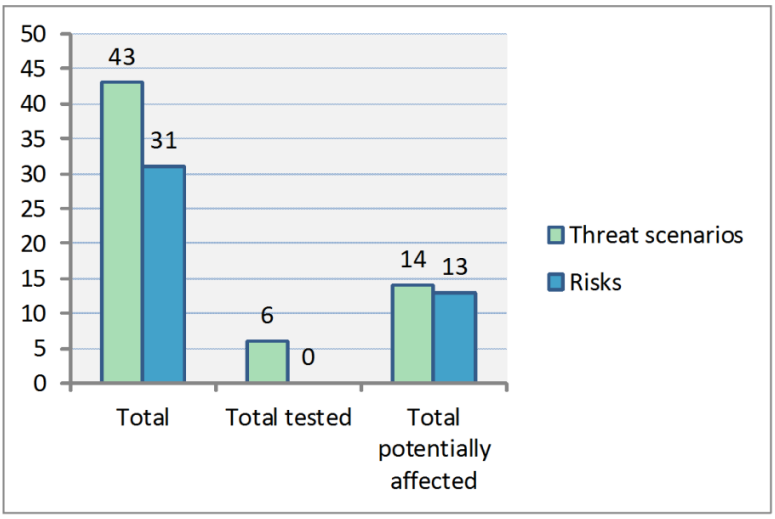
Figure 5. The number of risk model nodes, vulnerabilities, and relations before and after testing.



The only risk model elements that were tested in the case study were threat scenarios. The aim of testing a threat scenario was to discover whether the target of evaluation had vulnerabilities that could be exploited by the attack described by the threat scenario. Figure 6 shows the total number of threat scenarios and risks that were identified, the total number of tested threat scenarios, and the total number of unwanted incidents and threat scenarios *potentially affected* by the tested threat scenarios. All threat scenarios or unwanted incidents that a tested threat scenario *T* can lead up to are potentially affected by the testing. For example, if the likelihood value of *T* is edited after testing, then this may have a rippling effect on the likelihood values of all threat scenarios or unwanted incidents caused by *T*. It is therefore, in principle, possible that the complete risk model is affected by testing. However, this depends on which threat scenarios are selected for testing in the testing phase. As can be deduced from Figure 6, 6 out of 43 threat scenarios were tested, 14 out of 43 threat scenarios were potentially affected by the testing (this

includes the 6 threat scenarios selected for testing), and 13 out of 31 risks were potentially affected by the testing.

Figure 6. The number of risks and threat scenarios tested and potentially affected by the testing.



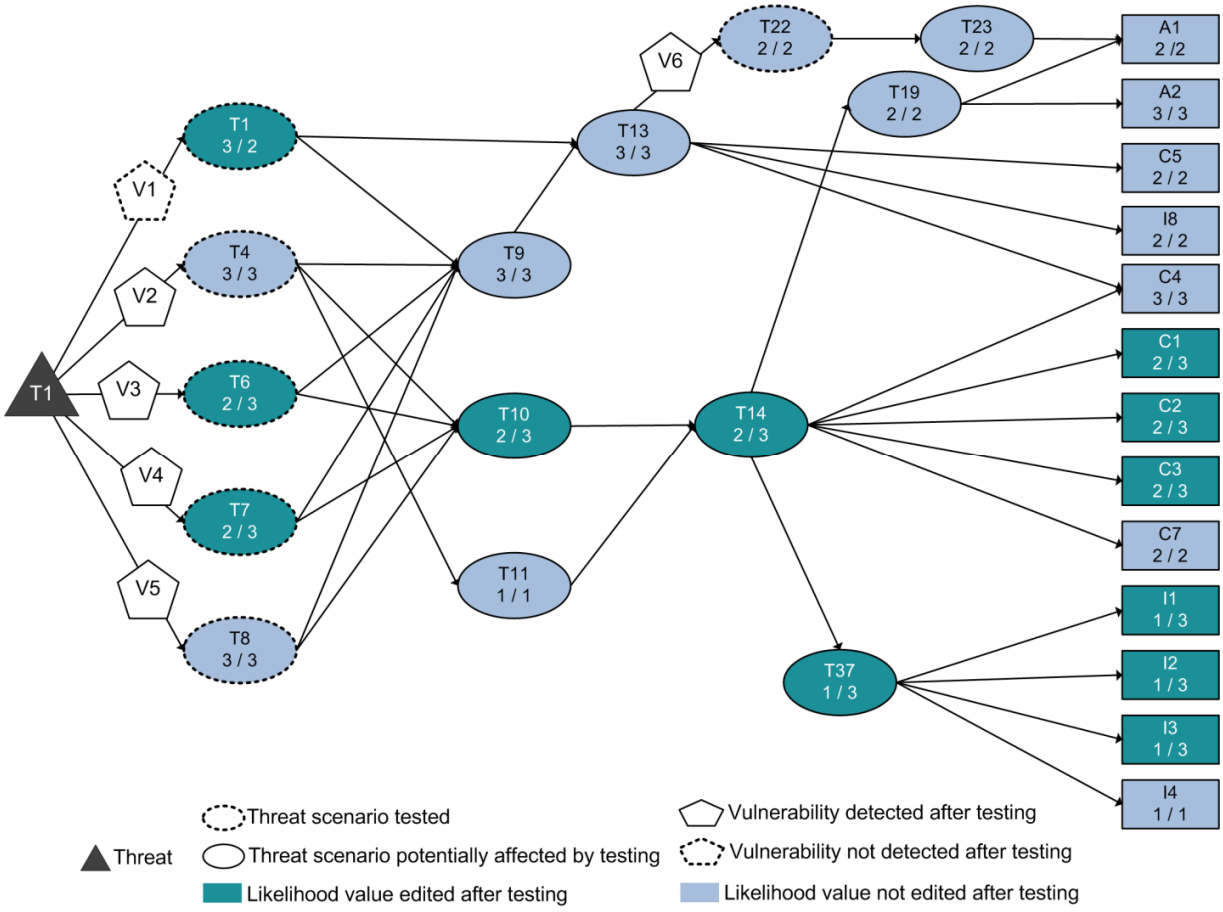
The risk model in Figure 7 represents an anonymized excerpt of the risk model identified in the case study. Furthermore, it shows all risk model elements that were tested and potentially affected by the testing, as well as the difference between the threat scenarios, risks, and vulnerabilities before and after testing. The threat scenarios that were tested are represented by ellipses with a dotted outline. All the other elements of the diagram are potentially affected by the tested threat scenarios. It can be noted that the level of indirection from the tested threat scenarios to the risks is quite large.

The vulnerabilities *V1* and *V6* were identified as potential vulnerabilities during the risk assessment, that is, prior to the testing. By testing threat scenario *T1*, we discovered that vulnerability *V1* did not exist and it was therefore removed from the risk model. The testing of threat scenario *T22*, however, revealed that vulnerability *V6* did in fact exist and therefore remained in the risk model after testing. Furthermore, by testing threat scenarios *T4*, *T6*, *T7* and *T8*, we discovered and added four previously unknown vulnerabilities to the risk model (*V2*, *V3*, *V4* and *V5*). This resulted in a total of five vulnerabilities in the risk model after testing.

The adding and deleting of vulnerabilities caused an update in the likelihood of some threat scenarios and risks. In Figure 7, each threat scenario and risk *TR* has a label of the form *i / j* which means that *TR* had likelihood value *i* before testing and *j* after the testing. The likelihood scale that was used in the case study can be mapped to a number between *1* and *5*, where *1* represents the most unlikely value and *5* represents the most likely value. All the threat scenarios and risks whose likelihood values were edited after testing are in Figure 7 represented with a darker color than the threat scenarios and risks that were not edited. Note that all except one risk model element (threat scenario *T1*) whose likelihood values were edited after testing were estimated to be more likely after testing than before testing. Threat scenario *T1* was estimated to be less likely after testing due to the absence of vulnerability *V1*.

Recall that risks may have risk levels which are calculated based on likelihood and consequence values. The change in likelihood values resulted in the risk levels of four risks being changed after testing; the risk level of risks *C2*, *I1*, *I2*, and *I3* were changed as a result of the increase in likelihood, while the risk level of risks *C1* and *C3* did not change although their likelihood increased after testing. More specifically, the risks *C2* and *I2* changed from *low* to *medium* with respect to risk level, while the risks *I1* and *I3* changed from *low* to *high*. The risk *C1* remained at risk level *low* after testing, while the risk *C3* remained at risk level *medium* after testing. The reason why the risk level of risks *C1* and *C3* did not increase was because the increase of their likelihood did not have sufficiently high impact to increase their risk level. The possible risk levels in the case studies were low, medium, and high.

Figure 7. Difference between risk models before and after testing.

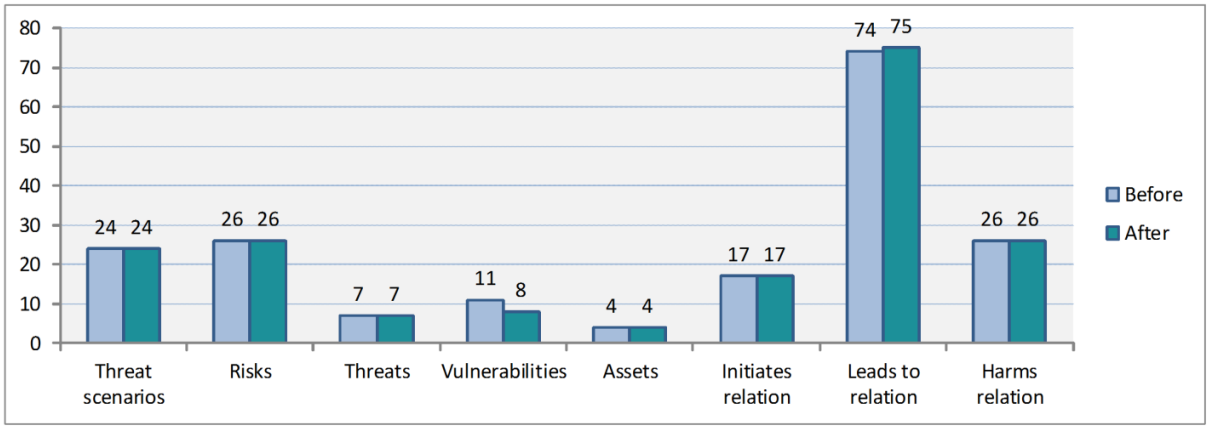


Results Obtained in Case Study 2

Similar to the first case study, no nodes were added to or deleted from the risk model after testing. One vulnerability was added to the risk model, while four vulnerabilities were deleted from the risk model after testing. In addition, one leads-to relation was added *as part of* the one

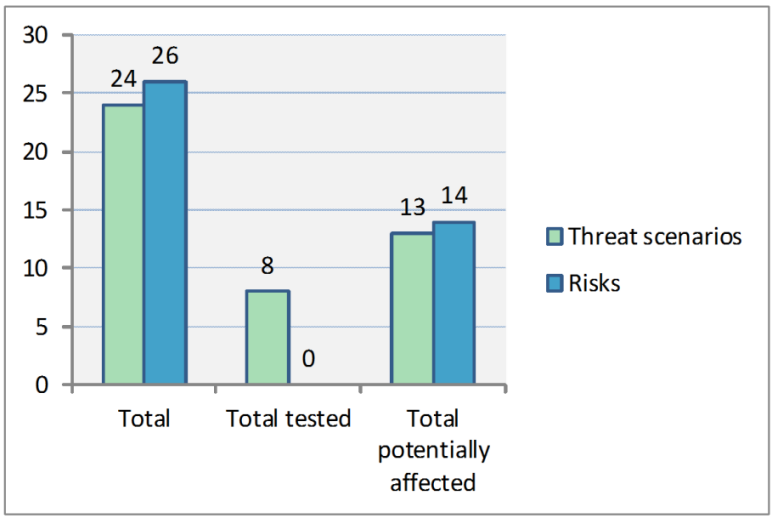
vulnerability added to the risk model. Figure 8 illustrates the number of risk model nodes, the number of vulnerabilities, and the number of relations before and after testing.

Figure 8. The number of risk model nodes, vulnerabilities, and relations before and after testing.



The only risk model elements that were tested in the second case study were, as in the first case study, threat scenarios. The chart in Figure 9 shows the total number of threat scenarios and risks that were identified, the total number of tested threat scenarios, and the total number of unwanted incidents and threat scenarios potentially affected by the tested threat scenarios. Furthermore, the chart shows that 8 out of 24 threat scenarios were tested, 13 out of 24 threat scenarios were potentially affected by the testing (this includes the 8 threat scenarios selected for testing), and 14 out of 26 risks were potentially affected by the testing.

Figure 9. The number of risks and threat scenarios tested and potentially affected by the testing.

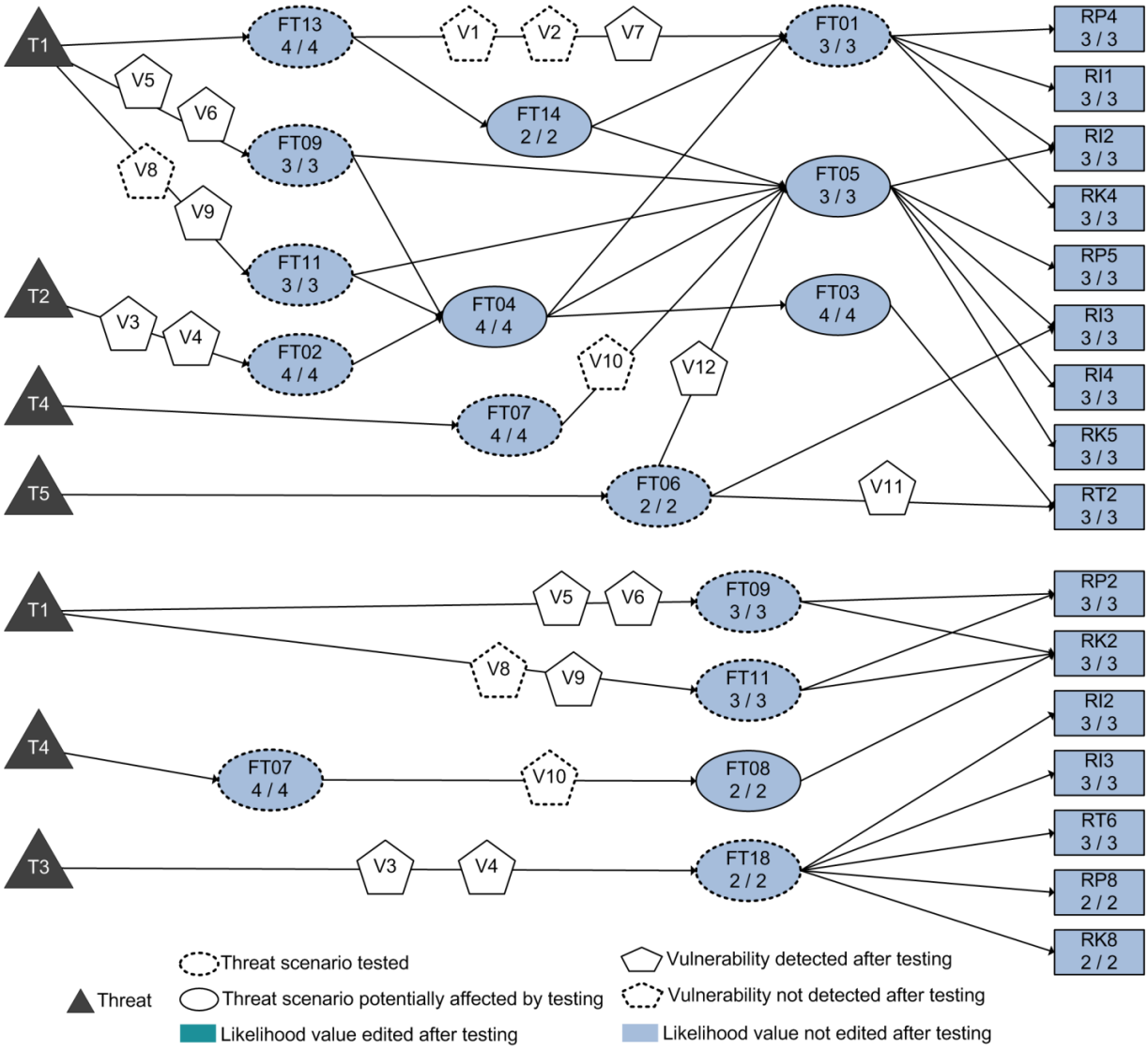


Similar to Figure 7, Figure 10 represents an anonymized excerpt of the risk model identified in the case study. The graphical notation of Figure 10 is equivalent to the notation used in Figure 7.

In the second case study, all vulnerabilities except one (*V11*) were modeled as potential vulnerabilities during the risk assessment. By testing the threat scenarios selected for testing, we discovered that the vulnerabilities *V1*, *V2*, *V8* and *V10* did not exist. These vulnerabilities were therefore removed from the risk model after testing. The testing revealed that the other seven presumed vulnerabilities did exist and therefore remained in the risk model after testing. In addition, by testing threat scenario *FT06* we discovered that *FT06* may also lead to risk *RT2* by exploiting a previously unknown vulnerability *V11*. Thus, we added vulnerability *V11* to the risk model. This resulted in a total of eight vulnerabilities in the risk model after testing.

However, the adding and deleting of vulnerabilities did not cause an update of the likelihood values. This is because the vulnerabilities were not sufficiently severe to affect the likelihood of the tested threat scenarios. As a result, none of the tested threat scenarios changed likelihood value after testing. Hence, none of the threat scenarios and risks potentially affected by the testing changed likelihood value.

Figure 10. Difference between risk models before and after testing.



DISCUSSION

In this section, we discuss the results obtained in the case studies with respect to our research questions.

To What Extent Are the Test Results Useful for Correcting the Risk Model by Adding Elements (RQ1)?

Based on the discussion in Section 5 and the numbers in Figure 5 and Figure 8, we know that new vulnerabilities were added to the risk models after testing. In addition, we see from Figure 8 that one new relation was added to the risk model after testing. However, this relation was added as part of a vulnerability. The relation would not been added if the vulnerability had not been discovered. No other kinds of risk model elements were added.

Why did the testing only yield new information about the vulnerabilities? The main reason for this is that the tests were designed with respect to the threat scenarios, and the purpose of the tests was to identify vulnerabilities that could be exploited by the threat scenarios. In other words, the tests were designed to uncover vulnerabilities; not unknown assets, threats, threat scenarios, or risks. These elements were instead part of the context in which the testing was performed.

Recall that an asset is something that is of value for the party, and that can be harmed by a risk. If a party has no assets, then there is no reason to conduct a risk analysis. For this reason, assets are always identified in the beginning of the risk analysis, before the risks are identified. In our experience, the process of identifying the risks has never led to the identification of new assets because the assets are then part of the context of the risk identification. The same is also true for the testing.

The argument is similar regarding threats. A threat is a potential cause of an unwanted incident such as a hacker, an insider or a virus, and the testing is performed *with regards to* the identified threats. It therefore seems unlikely that the testing would uncover additional threats.

In principle, we cannot rule out that testing could yield information that would lead to the identification of new threat scenarios or risks. For instance, it might be the case that a threat scenario may be refined (that is, split up into more than one threat scenario) after testing, or lead to the identification of an unwanted incident that had not been previously thought of. However, as long as the tests are designed to uncover vulnerabilities, we believe that this would be unlikely.

As mentioned in Section 5, some vulnerabilities in the case studies were only discovered by testing, and were not known to us prior to the testing. That is, they were not identified during the risk assessment phase, but only during the testing phase. It is worth noticing that these vulnerabilities could never have been uncovered if we had performed risk analysis alone (without doing the testing), regardless of how much effort we would have spent. This is because of the limited amount of information that was available to us in the target description on which the risk assessment was based.

To What Extent Are the Test Results Useful for Correcting the Risk Model by Deleting Elements (RQ2)?

Based on the discussion in Section 5, we know that the test results led to the deletion of exactly one kind of risk model element, namely vulnerabilities. In the first case study, the test results led to the deletion of one vulnerability, while in the second case study four vulnerabilities were deleted after testing. We believe that this result is generalizable. That is, in general, threats, threat scenarios, risks and assets are unlikely to be deleted after testing, whereas vulnerabilities may be deleted.

The reason why we deleted the vulnerabilities after testing was that the testing provided evidence that a potential vulnerability identified in the risk assessment phase was actually not present in the system. Based on this, as well as expert judgement, we removed the vulnerabilities from the risk models. We also believe that in general, testing can support the deletion of vulnerabilities,

since the tests can be designed to check whether a specific vulnerability is actually present in the system or not.

The reason why threats and assets are unlikely to be deleted after testing is the same as for why they are unlikely to be added after testing. That is, the assets and threats are part of the context in which the testing is performed, and the testing is therefore unlikely to yield information about this context.

As for threat scenarios and unwanted incidents, these are risk model elements that contain assigned likelihood values. Therefore, there will never be a need to delete these from the risk model after testing. Instead of deleting them from the risk model, we would assign a low likelihood value on these risk model elements.

To What Extent Are the Test Results Useful for Correcting the Risk Model by Editing Elements (RQ3)?

The results obtained for the first case study show that 6 out of 43 threat scenarios and 6 out of 31 risks were edited with respect to likelihood values. The six threat scenarios and the six risks are illustrated with darker color in Figure 7. Moreover, 4 out of the 6 risks that were edited with respect to likelihood values were assigned a higher risk level as a result of the new likelihood values. The results obtained for the second case study show that there were no change in the likelihood values, because the identified vulnerabilities in the second case study were not sufficiently severe to have an influence on the likelihood values.

For all risk model elements that were edited in the first case study (with the exception of one), the likelihood value was increased after testing, that is, they were believed to be more likely after testing than before testing. The reason for this was that the testing uncovered vulnerabilities that were previously unknown, which led to the belief that certain threat scenarios and risks were more likely to occur than believed before testing.

In one of the threat scenarios in the first case study, the likelihood value was decreased after testing as a result of deleting one vulnerability. Although four vulnerabilities were removed after testing in the second case study, they were not sufficiently severe to decrease the likelihood values of the threat scenarios.

In general, we believe that testing will uncover information that may cause the likelihood values of threat scenarios and unwanted incidents to be edited after testing. The testing did not result in editing the consequence values that unwanted incidents have on assets. The reason for this is that all the tests were designed to uncover information about vulnerabilities that would increase or decrease the likelihood of a successful attack. The consequence of a successful attack was already known in advance. We believe this result is generalizable. As long as all risks have been identified before testing, their consequences can be estimated before testing, and it is unlikely that the testing will uncover information which will cause the consequence values to change.

Summary and Lessons Learned

Our overall hypothesis is that security testing is useful for validating and correcting the security risk models. Based on our experience, we see that the test results are useful for **correcting** the risk models in terms of *adding* or *deleting* vulnerabilities, as well as *editing* likelihood values.

Vulnerabilities are either added to or deleted from the risk models because, in our approach, the test cases are designed to discover vulnerabilities. However, even if it is unlikely, we cannot completely rule out the possibility of identifying new threat scenarios or risks based on the test results. Furthermore, the test cases are designed with respect to threat scenarios which contain likelihood values. The adding or deleting of vulnerabilities may therefore have an influence on the likelihood of the corresponding threat scenarios. Thus, adding or deleting vulnerabilities may result in editing likelihood values.

We have also seen that the test results are useful for **validating** the risk models in terms of discovering the presence or absence of presumed vulnerabilities, and thereby increasing the trust in the risk models. Even if the testing does not identify new vulnerabilities, or lead to an update of likelihood values, it is still useful for validating the correctness of the risk models. For example, in the second case study, the testing did not lead to any update of the likelihood values, and it confirmed the presence of 7 out of 11 presumed vulnerabilities. This indicates that the risk assessment was of high quality.

In our experience, an industrial security risk analysis without testing is typically carried out between 250 and 300 hours. In the first case study, we spent approximately 293 hours, and in the second case study we spent approximately 322 hour. Furthermore, in both case studies, approximately 25% of the total time was devoted to the testing phase. We believe that the return of investment in testing outweighs the alternative, which is carrying out the risk analysis without testing. We believe this because, in both case studies, testing helped us in validating and correcting the risk models as described above, and thereby mitigated the uncertainty in the risk models. In both case studies, it would not have been possible to validate or correct the risk picture without test execution. This is due to the limited amount of information that was available to us in the target description, and because the testing uncovered issues which only appeared in extremely specific circumstances which could not have been reproduced without executing the system under analysis.

RELATED WORK

In this paper, we have evaluated a test-driven risk analysis approach specialized on security. Test-driven risk analysis is the process of using testing to improve the risk analysis results. We distinguish this from the notion of risk-driven (or risk-based) testing which is the process of using risk analysis to improve the test process. This distinction is usually unclear in the literature as argued in (Erdogan, Li, Runde, Seehusen, & Stølen, 2014). However, most of the literature on combining risk analysis and testing fits into the latter category (risk-driven testing). Indeed, the idea of combining risk analysis and testing first originated from the testing community, and this integration is also reflected in the recent software testing standard ISO/IEC/IEEE 29119 (International Organization for Standardization, 2013). A number of risk-driven testing approaches have been proposed over the years. These approaches have in common that the identified risks become the driving factor for one or more of the activities within a testing

process: Test planning, test design, test implementation, test execution, and test evaluation (Felderer & Schieferdecker, 2014).

Based on our systematic literature review on approaches for the combined use of risk analysis and testing (Erdogan et al., 2014) we identified three other approaches that address test-driven risk analysis. The first two approaches described below do not address security. Nevertheless, they suggest techniques in which testing is used as a means to improve the risk analysis results.

Wong et al. (2005) suggests a test-driven risk analysis approach in which risk of code is described as the likelihood that a given function or block within source code contains a fault. Their mathematical risk model is updated based on metrics related to both the static structure of code as well as dynamic test coverage. A more complex static structure leads to higher risk, while more thoroughly tested code has less risk. The approach is evaluated on an industrial software used for configuring antennas. The approach is supported by a tool in which the only feedback to the risk analyst or developer is a categorization of risky source code.

Schneidewind (2007) suggests an approach that supports both risk-driven testing and test-driven risk analysis. With a focus on reliability, the approach suggests a risk-driven reliability model and testing process where the risk of software failure is used to drive test scenarios and reliability predictions. Both consumer and producer risks are taken into account. In addition to comparing empirical values of risk and reliability to specified threshhold values, emphasis is placed on evaluating the mathematical model that predicts risk and reliability. The evaluation is carried out in terms of an hypotehtical example problem applied on the NASA shuttle flight software.

Großmann et al. (2014a, 2014b) suggest an approach that supports both risk-driven security testing and test-driven security risk analysis. Their approach is somewhat similar to our approach in the sense that it makes use of threat scenarios to identify security tests. This is the risk-driven testing part of their approach. Then, based on the test results, they update the risk models. This is the test-driven risk analysis part of their approach. However, although it is mentioned that the test results may lead to the identification of additional vulnerabilities, threat scenarios, and unwanted incidents, it is not explained exactly how this may be achieved. In particular, as we have discussed, it is not obvious how additional threat scenarios or unwanted incidents can be identified by testing threat scenarios. Moreover, they mention that based on the test results, the likelihood values may become more precise. Again, it is not clear how this is achieved, and this is also not a straight forward process. For example, in our approach, we assign a level of severity to the identified vulnerabilities. Then, based on the severity level, we assess whether the likelihood value of the relevant threat scenario should be increased, decreased or remain unchanged. The approach is supported by a tool. Großmann et al. (2014b) gives an analytical evaluation of the tool, and mention that the approach will be evaluated in future case studies.

CONCLUSION

We have described an evaluation of a process for test-driven security risk analysis based on our experiences from applying this process in two industrial case studies. The objective of the evaluation was to evaluate how useful testing is for validating and correcting the risk models produced in the security risk assessment phase. To make the evaluation precise, we analyzed the difference between the risk models produced before and after the testing.

The process of testing yielded information which led to an update of the risk models created before testing. Specifically, in the first case study, 4 vulnerabilities were added to the risk model, and 1 vulnerability was deleted from the risk model. This resulted in a corrected risk model containing 5 vulnerabilities instead of 2 vulnerabilities. In the second case study, 1 vulnerability was added to the risk model, while 4 vulnerabilities were deleted. This resulted in a corrected risk model containing 8 vulnerabilities instead of 11 vulnerabilities.

Furthermore, in the first case study, 6 out of 43 threat scenarios and 6 out of 31 risks were edited with respect to likelihood values. Moreover, 4 out of the 6 risks that were edited with respect to likelihood values were assigned a higher risk level as a result of the new likelihood values. In the second case study, on the other hand, there were no editing in terms of updating the likelihood values of the threat scenarios and unwanted incidents. The reason to this is that the vulnerabilities in the second case study were not sufficiently severe to affect the likelihood values. However, the fact that 7 out of 11 presumed vulnerabilities were confirmed present, and that there were no need in updating the likelihood values, suggests that the risk assessment was of high quality.

The testing was useful in the sense that it yielded more accurate risk models, which indicates that testing is indeed useful for validating and correcting risk models. Another important point is that the testing uncovered vulnerabilities that would never have been uncovered in the risk analysis phase, regardless of how much effort we would have spent in the risk analysis phase. This is because of the limited amount of information that was available to us in the target description on which the risk assessment was based. In other words, if the risk analysis phase had been extended with the effort spent on testing, we would not have uncovered the vulnerabilities that were uncovered in the testing phase.

ACKNOWLEDGMENTS

This work has been conducted as a part of the DIAMONDS project (201579/S10) and the AGRA project (236657) funded by the Research Council of Norway, as well as the RASEN project (316853) and the NESSoS network of excellence (256980) funded by the European Commission within the 7th Framework Programme. The first version of this paper was presented in the International Workshop on Quantitative Aspects in Security Assurance (QASA 2012).

REFERENCES

CORAS. (2015). *The CORAS Tool*. Retrieved January 27, 2015, from http://coras.sourceforge.net/coras_tool.html

Eclipse. (2015). *Eclipse*. Retrieved January 27, 2015, from https://eclipse.org/home/index.php

Erdogan, G., Li, Y., Runde, R. K., Seehusen, F., & Stølen, K. (2014). Approaches for the combined use of risk analysis and testing: A systematic literature review. *International Journal on Software Tools for Technology Transfer, 16*(5), 627-642.

Felderer, M., & Schieferdecker, I. (2014). A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer, 16*(5), 559-568.

Google. (2015). *Chrome*. Retrieved January 27, 2015, from http://www.google.com/chrome/

Großmann, J., Berger, M., & Viehmann, J. (2014a). A trace management platform for risk-based security testing. *In 1st International Workshop on Risk Assessment and Risk-driven Testing (RISK'13)* (pp. 120-135). International Publishing Switzerland: Springer.

Großmann, J., Schneider, M., Viehmann, J., & Wendland, M.-F. (2014b). Combining risk analysis and security testing. *In 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14)* (pp. 322-336). Berlin Heidelberg: Springer.

IBM. (2015). *IBM Rational Software Architect*. Retrieved January 27, 2015, from http://www.ibm.com/developerworks/downloads/r/architect/index.html

International Organization for Standardization. (2013). *ISO/IEC/IEEE 29119-1:2013(E), Software and system engineering - Software testing - Part 1: Concepts and definitions.* Geneva: International Organization for Standardization.

Lund, M. S., Solhaug, B., & Stølen, K. (2011). *Model-Driven Risk Analysis - The CORAS Approach.* Berlin Heidelberg: Springer.

Mozilla. (2015, January 30). *Firefox Web browser*. Retrieved from https://www.mozilla.org/en-US/firefox/desktop/

Open Web Application Security Project. (2015). *OWASP Zed Attack Proxy Project*. Retrieved January 27, 2015, from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Portswigger. (2015). *Portswigger Burp Suite Free Edition*. Retrieved January 27, 2015, from http://portswigger.net/burp/download.html

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). *Case Study Research in Software Engineering: Guidelines and Examples.* Hoboken: John Wiley & Sons.

Schneidewind, N. F. (2007). Risk-driven software testing and reliability. *International Journal of Reliability, Quality and Safety Engineering, 14*(2), 99-132.

Seehusen, F. (2014). A technique for risk-based test procedure identification, prioritization and selection. *In 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14)* (pp. 277-291). Berlin Heidelberg: Springer.

Selenium. (2015). *SeleniumHQ*. Retrieved January 27, 2015, from http://www.seleniumhq.org/

Smartesting. (2015). *Smartesting CertifyIt*. Retrieved January 27, 2015, from http://www.smartesting.com/en/certifyit/

Wireshark. (2015). *Wireshark*. Retrieved January 27, 2015, from https://www.wireshark.org/

Wong, W. E., Qi, Y., & Cooper, K. (2005). Source code-based software risk assessing. *In Proceedings of the 2005 ACM Symposium on Applied Computing (SAC'05)* (pp. 1485-1490). New York: ACM.