SINTEF

SINTEF A27097 - Unrestricted

Report

Evaluation of the CORAL Approach for Risk-Driven Security Testing Based on an Industrial Case Study

Author(s)

Gencer Erdogan[1], Ketil Stølen[1], and Jan Øyvind Aagedal[2]

[1] Department for Networked Systems and Services, SINTEF ICT, PO Box 124 Blindern, N-0314 Oslo, Norway

[2] Accurate Equity, Martin Linges vei 25, N-1364 Fornebu, Norway

SINTEF ICT

Networked Systems and Services

2015-07-28

SINTEF

SINTEF IKT
SINTEF ICT

Address:
Postboks 124 Blindern
NO-0314 Oslo
NORWAY

Telephone:+47 73593000
Telefax:+47 22067350

postmottak.IKT@sintef.no
www.sintef.no
Enterprise /VAT No:
NO 948 007 029 MVA

KEYWORDS:
Case study,
Risk-driven,
Risk-based,
Security testing

Report

Evaluation of the CORAL Approach for Risk-Driven Security Testing Based on an Industrial Case Study

VERSION
Final

DATE
2015-07-28

AUTHOR(S)
Gencer Erdogan, Ketil Stølen, and Jan Øyvind Aagedal

CLIENT(S)
Norwegian Research Council

CLIENT'S REF.
201579/S10

PROJECT NO.
102002253

NUMBER OF PAGES/APPENDICES:
20/0

ABSTRACT
The CORAL approach is a model-based method to security testing employing risk assessment to help security testers select and design test cases based on the available risk picture. In this report we present experiences from using CORAL in an industrial case. The results indicate that CORAL supports security testers in producing risk models that are valid and threat scenarios that are directly testable. This, in turn, helps testers to select and design test cases according to the most severe security risks posed on the system under test.

PREPARED BY
Gencer Erdogan

SIGNATURE
Gencer Erdogan

CHECKED BY
Atle Refsdal

SIGNATURE
Atle Refsdal

APPROVED BY
Fredrik Seehusen

SIGNATURE
Fred Se

REPORT NO.	ISBN	CLASSIFICATION	CLASSIFICATION THIS PAGE
SINTEF A27097	978-82-14-05907-6	Unrestricted	Unrestricted

Table of Contents

1 Introduction

Security testers face the problem of determining the tests that are most likely to reveal severe security vulnerabilities. Risk-driven security testing has been proposed in response to this challenge. Potter and McGraw [20] argue that security testers must use a risk-driven approach to security testing, because by identifying risks in the system and creating tests driven by those risks, a security tester can properly focus on aspects of the system in which an attack is likely to succeed. Unfortunately, only a handful approaches to risk-driven testing specifically address security, and the field is immature and needs more formality and preciseness [4].

We have developed a method for risk-driven security testing supported by a domain-specific language which we refer to as the CORAL approach, or just CORAL [5, 6]. It aims to help security testers to select and design test cases based on the available risk picture.

In this report we present experiences from applying CORAL in an industrial case. We evaluate to what extent the CORAL approach helps security testers in selecting and designing test cases. The system under test is a comprehensive web-based e-business application designed to deliver streamlined administration and reporting of all forms of equity-based compensation plans, and is used by a large number of customers across Europe. The system owner, which is also the party that commissioned the case study (often referred to as party in the following), require full confidentiality. The results presented in this report are therefore limited to the experiences from applying the CORAL approach.

The report is organized as follows. In Sect. 2 we give an overview the CORAL approach. In Sect. 3 we present our research method. In Sect. 4 we give an overview of the case study, and in Sect. 5 we present the obtained results organized according to our research questions. In Sect. 6 we discuss these results, and finally in Sect. 7 we relate our work to other approaches and conclude by highlighting key findings.

2 The CORAL Approach

The CORAL approach consists of a domain-specific risk analysis language and a method for risk-driven security testing within which the language is tightly integrated. Figure 1 presents an example of a risk model expressed in the CORAL language. The dashed arrows are not part of the model, but used to point out the various constructs explained below.

A *threat* is a potential cause of an unwanted incident. A *threat scenario* is a chain or series of events that is initiated by a threat and that may lead to an unwanted incident. An *asset* is something to which the party on whose behalf we are testing assigns value and hence for which the party requires protection. A *new message* is a message initiated by a threat, and is represented by a red triangle at the transmission end. An *altered message* is a message in the system under test (SUT) that has been altered by a threat to deviate from its expected behavior.

It is represented by a triangle with red borders and white fill. A *deleted message* is a message in the SUT that has been deleted by a threat. It is represented by a triangle with red borders and a red cross in the middle. An *unwanted incident* is a message modeling that an asset is harmed or its value is reduced. It is represented by a yellow explosion sign. A *frequency* is the frequency of either the transmission or the reception of a message. A *conditional ratio* is the ratio by which a message is received, given that it is transmitted. A *consequence* is the consequence an unwanted incident has on an asset. A risk, in our approach, is the frequency of an unwanted incident and its consequence for a specific asset.

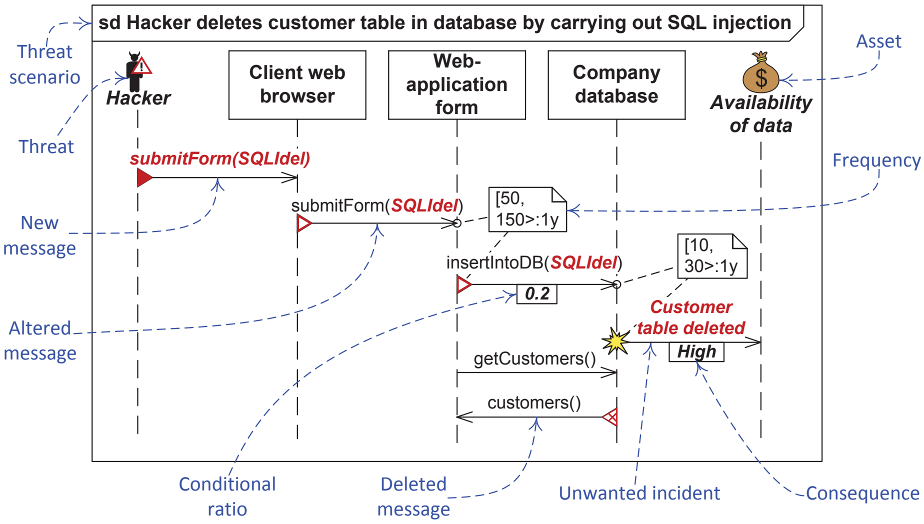

Fig. 1. Example of a CORAL risk model.

As illustrated in Fig. 2, the CORAL method expects a description of the SUT as input. The description may be in the form of system diagrams and models, use case documentation, source code, executable versions of the system, and so on. The CORAL method involves seven steps grouped into three phases: Test planning, security risk assessment, and security testing. The output from applying CORAL is a test report.

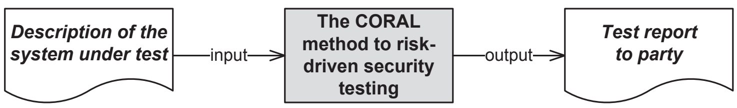

Fig. 2. Input and output of the CORAL method.

In Phase 1, we identify security assets to be protected, define frequency and consequence scales, and define the risk evaluation matrix based on the frequency and consequence scales.

In Phase 2, we carry out the risk modeling in three consecutive steps. First, we identify security risks by analyzing the models of the SUT with respect to the security assets. Then we identify threat scenarios that may cause the security risks. Second, we estimate the frequency of the identified threat scenarios and risks by making use of the frequency scale, and the consequence of risks by making use of the consequence scale. Third, we evaluate the risks with respect to their frequency and consequence estimates and select the most severe risks to test.

In Phase 3, we conduct security testing in three consecutive steps. First, for each risk selected for testing, we identify the threat scenario in which the risk occurs and specify a test objective for that threat scenario. To obtain a test case fulfilling the test objective, we use stereotypes from the UML Testing Profile [17] to annotate the threat scenario with respect to the test objective. Second, we carry out security testing with respect to the test cases. The test cases may be executed manually, semi automatically, or automatically. Third, based on the test results, we write a test report.

3 Research Method

As illustrated by Fig. 3 we conducted the case study in four main steps. First, we designed the case study by defining the objective, the units of analysis, as well as the research questions. Second, we carried out the CORAL approach within an industrial setting. Third, we collected the relevant data produced by executing the CORAL approach. Fourth, we analyzed the collected data with respect to our research questions. This research approach is inspired by the guidelines for case study research in software engineering provided by Runeson et al. [22].

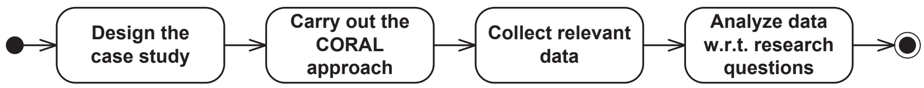

Fig. 3. The main activities of the research method.

As pointed out in Sect. 1, the objective of the case study was to evaluate to what extent the CORAL approach helps security testers in selecting and designing test cases. The test report delivered to the party that commissioned the case study describes, in addition to the test results, risk models and security tests designed with respect to the risk models. Our hypothesis was that the report is good in the sense that (1) the risk models are valid, and (2) the threat scenarios are directly testable. By a directly testable threat scenario, we mean a threat scenario that can be reused and specified as a test case based on its interactions. Thus, the units of analysis in this case study are the risk models.

With respect to point (1), we defined two research questions (RQ1 and RQ2). With respect to point (2), we defined one research question (RQ3). Additionally, we carried out both black-box and white-box testing of the SUT, because we were interested in investigating the usefulness of the CORAL approach for both black-box and white-box testing (RQ4).

RQ1 To what extent is the risk level of identified risks correct?

RQ2 To what extent are relevant risks identified compared to previous penetration tests?

RQ3 To what extent are the threat scenarios that causes the identified risks directly testable?

RQ4 To what extent is the CORAL approach useful for black-box testing and white-box testing, respectively?

4 Overview of Industrial Case Study

As mentioned in Sect. 1, the system under test was a web-based application providing services related to equity-based compensation plans. The web application was deployed on the servers of a third party service provider and 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 various features of the web application.

In order to limit the scope of the testing, we decided to test two features available to customers: a feature for selling shares (named Sell Shares), and a feature for exercising options for the purpose of buying shares in a company (named Exercise Options). In the following, we explain how we carried out the CORAL approach by taking you through a fragment of the case study. We consider only Exercise Options and two potential threat scenarios: one from a black-box perspective and one from a white-box perspective.

4.1 Test Planning (Phase 1)

We modeled Exercise Options from a black-box perspective by observing its behavior. That is, we executed Exercise Options using a web browser, observed its behavior, and created the model based on that. We also modeled Exercise Options from a white-box perspective by executing and analyzing its source code. Figures 4a and 4b show the black-box model and the white-box model of Exercise Options, respectively.

Together with the party we decided not to consider security risks related to infrastructure because this was a contractual responsibility of the service provider hosting the web application. Instead, we focused on security risks that may be

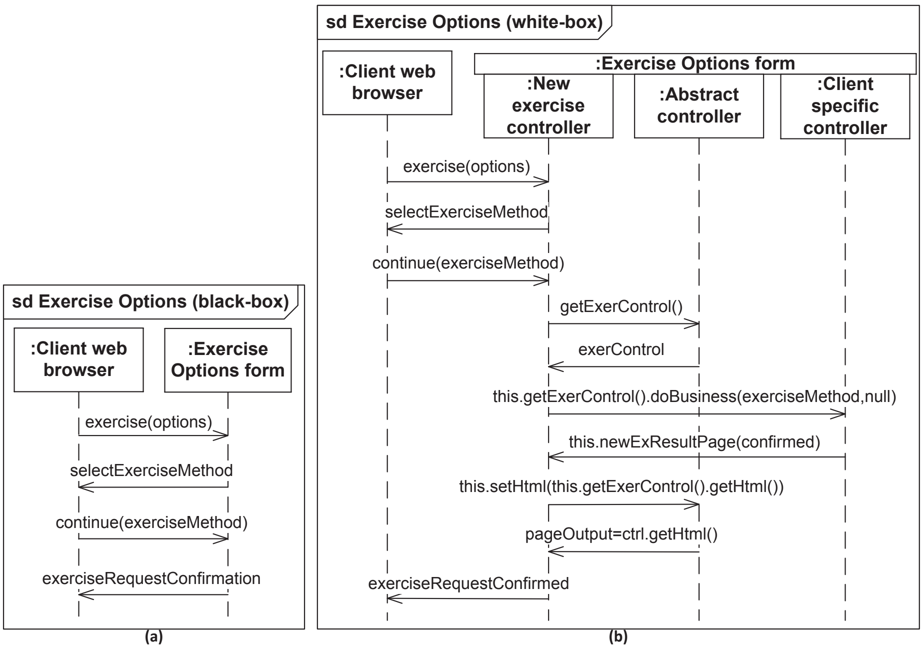

Fig. 4. (a) Black-box model of feature Exercise Options. **(b)** White-box model of feature Exercise Options.

introduced via the application layer. Thus, the threat profile is someone who has access to Exercise Options, but who resides outside the network boundaries of the service provider. We identified security assets by consulting the party. The security asset identified for Exercise Options was *integrity of data*.

We also defined a frequency scale and a consequence scale together with the party. The frequency scale consisted of five values (Certain, Likely, Possible, Unlikely, and Rare), where each value was defined as a frequency interval. For example, the frequency interval for likelihood Possible was *[5,20⟩:1y*, which means "from and including 5 to less than 20 times per year." The consequence scale also consisted of five values (Catastrophic, Major, Moderate, Minor, and Insignificant), where each value described the impact by which the security asset is harmed. For example, consequence Major with respect to security asset *integrity of data* was defined as "the integrity of customer shares is compromised." The scales were also used to construct the risk evaluation matrix illustrated in Fig. 7.

4.2 Security Risk Assessment (Phase 2)

We identified security risks by analyzing the black-box and white-box models of Exercise Options with respect to security asset *integrity of data*. We did this by first identifying unwanted incidents that have an impact on the security asset.

Second, we identified alterations in the messages that had to take place in order to cause the unwanted incidents (to be represented as altered messages). Third, we identified messages initiated by the threat which in turn could cause the alterations (to be represented as new messages).

Let us consider a threat scenario for the black-box model of Exercise Options. Assume that a malicious user attempts to access another system feature, say an administrative functionality, by altering certain parameters in the HTTP request sent to Exercise Options. The malicious user could achieve this, for example, by first intercepting the request containing the message *continue(exerciseMethod)* using a network proxy tool such as OWASP ZAP [19], and then altering the parameter *exerciseMethod* in the message. This alteration, could in turn give the malicious user access to another system feature. This unwanted incident occurs if the alteration is successfully carried out, and Exercise Options responds with another system feature instead of the expected message *exerciseRequestConfirmation*. Thus, the unwanted incident may occur after the reception of the last message in the black-box model (Fig. 4a). The resulting threat scenario is shown in Fig. 5. We carried out a similar analysis during white-box testing by analyzing the model in Fig. 4b. The resulting threat scenario for the white-box model is shown in Fig. 6.

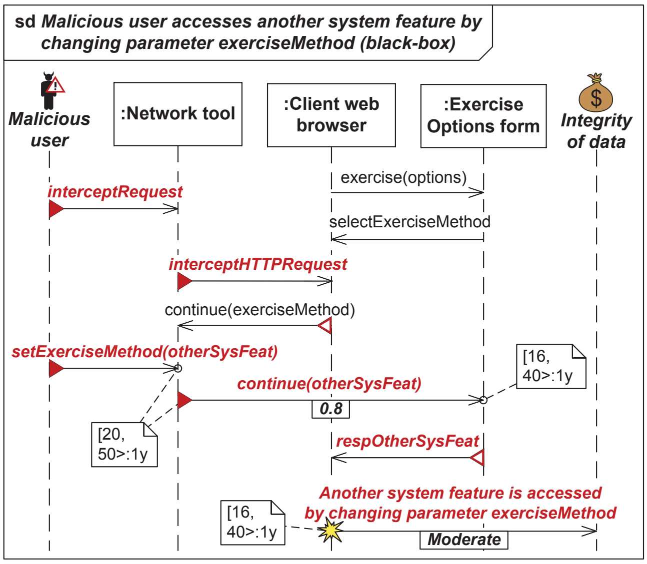

Fig. 5. A threat scenario for the black-box model of feature Exercise Options.

In order to estimate how often threat scenarios may occur, in terms of frequency, we based ourselves on knowledge data bases such as OWASP [18], reports and papers published within the software security community, as well as expert knowledge within security testing. We see from Fig. 5 that the malicious

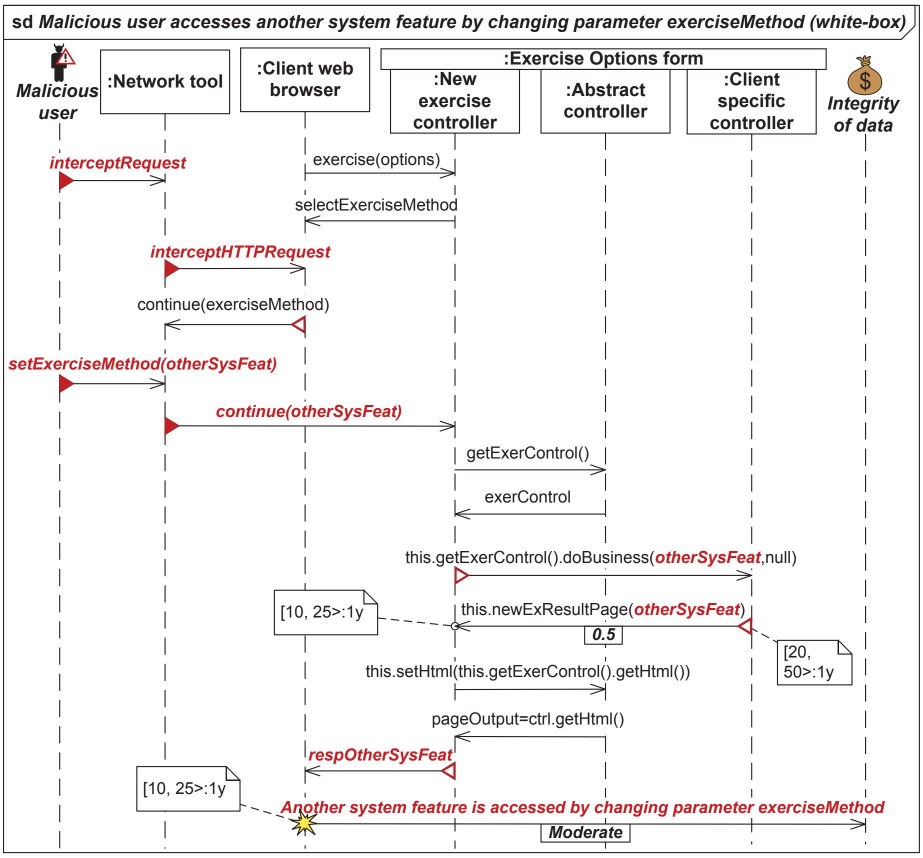

Fig. 6. A threat scenario for the white-box model of feature Exercise Options.

user successfully alters the parameter *exerciseMethod* with frequency *[20,50⟩:1y*. Given that parameter *exerciseMethod* is successfully altered and transmitted, it will be received by Exercise Options with conditional ratio *0.8*. The conditional ratio causes the new frequency *[16,40⟩:1y* for the reception of message *continue(otherSysFeat)*. This is calculated by multiplying *[20,50⟩:1y* with *0.8*. Given that message *continue(otherSysFeat)* is processed by Exercise Options, it will respond with another system feature. This, in turn, causes the unwanted incident (security risk) to occur with frequency *[16,40⟩:1y*. The unwanted incident has an impact on security asset *integrity of data* with consequence *Moderate*.

Figure 7 shows the obtained risk evaluation matrix. The numbers in the matrix represent the 21 risks identified in the case study. Each risk was plotted in the matrix according to its frequency and consequence estimate. Risks are grouped in nine levels horizontally on the matrix where Risk Level 1 is the lowest risk level and Risk Level 9 is the highest risk level. The risk level of a risk is identified by mapping the underlying color to the column on the left-hand side of the matrix. For example, Risks 11 and 19 have Risk Level 8, while Risk 20 has Risk Level 4. The risk aggregation did not lead to an increase in risk level for any of the risks. The suspension criterion in this case study was defined as "test all risks of Risk Level 6 or more." Based on this criterion, we selected 11 risks to test from the risk evaluation matrix.

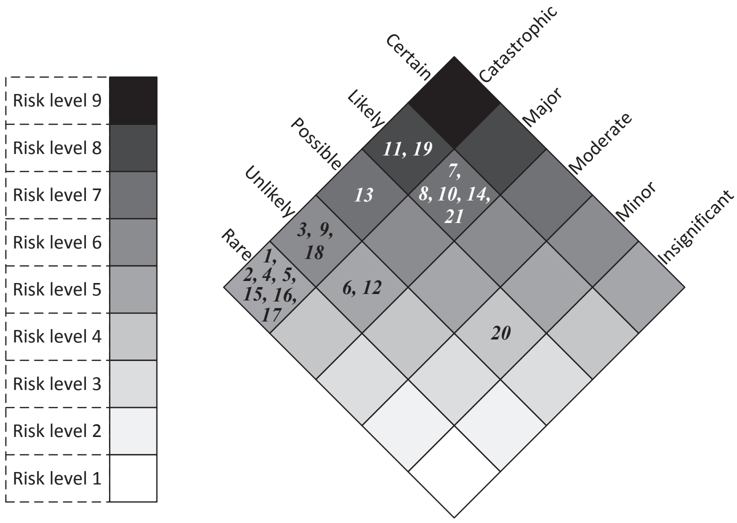

Fig. 7. Risk evaluation matrix.

4.3 Security Testing (Phase 3)

The test objective for the threat scenarios in Figs. 5 and 6 was defined as: "Verify whether the malicious user is able to access another system feature by changing parameter *exerciseMethod* into a valid system parameter". Based on this test objective, we annotated the threat scenarios with stereotypes from the UML testing profile [17]. For example, the resulting security test case for the threat scenario in Fig. 5 is shown in Fig. 8. Needless to say, the security tester takes the role as "malicious user" in the test case.

We carried out all black-box tests manually and used the OWASP Zed Attack Proxy tool [19] to intercept the HTTP requests and responses. We carried out all white-box tests semi automatically supported by the debug mode in Eclipse IDE, which was integrated with a web-server and a database. We also carried out automatic source code review using static source code analysis tools for the purpose of identifying potential threat scenarios. The tools we used for this purpose were Find Security Bugs V1.2.1 [7], Lapse plus V2.8.1 [13], and Visual Code Grepper (VCG) V2.0.0 [24].

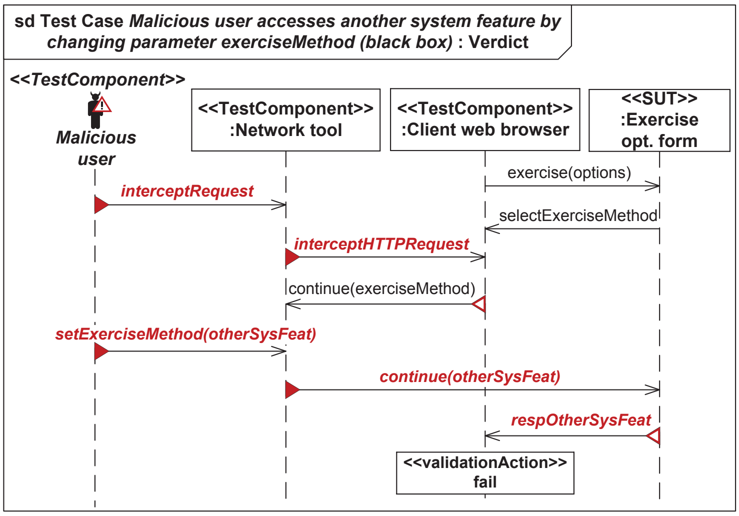

Fig. 8. Security test case based on the threat scenario in Fig. 5.

5 Case Study Results

In this section we present the results from the case study. We group the results with respect to our research questions.

RQ1 To what extent is the risk level of identified risks correct? As shown in the risk evaluation matrix in Fig. 7, we identified in total 21 security risks. The risk aggregation did not lead to an increase in risk level for any of the risks. Based on the suspension criterion defined by the party, we tested 11 risks (all risks of Risk Level 6 or more).

The testing of these 11 risks revealed 11 vulnerabilities. The vulnerabilities were assigned a severity level based on a scale of three values (Low, Medium, and High). High severity means that the vulnerability should be treated as soon as possible, and one should consider taking the system offline while treating the vulnerability ("show stopper"). Medium severity means that the vulnerability is serious and should be treated based on available time and resources. Low severity means that the vulnerability is not serious and it should be treated if seen necessary. Four vulnerabilities were assigned severity Medium, and the remaining 7 vulnerabilities were assigned severity Low.

We also tested the 10 risks initially not selected for testing, in order to have a basis for comparison. This testing revealed only 2 vulnerabilities of severity Low.

RQ2 To what extent are relevant risks identified compared to previous penetration tests? The party commissioning the case study had previously executed commercial penetration tests. We did not get access to the reports or results from these penetration tests due to confidentiality reasons. However, it was confirmed by the party that we had identified 16 security risks which had also been identified by the previous penetration tests. Additionally, we had identified 5 new security risks which had not been identified by the previous penetration tests. Moreover, the party also confirmed that we had not left out any risks of relevance for the features considered in the case study.

RQ3 To what extent are the threat scenarios that causes the identified risks directly testable? The identified 21 security risks were caused by 31 threat scenarios. The 11 risks initially selected for testing were caused by 18 threat scenarios, while the remaining 10 risks were caused by 13 threat scenarios. We identified 18 security test cases based on the 18 threat scenarios causing the 11 risks. Similarly, we identified 13 security test cases based on the 13 threat scenarios causing the remaining 10 risks.

RQ4 To what extent is the CORAL approach useful for black-box testing and white-box testing, respectively? Table 1 gives an overview of the results obtained during black-box testing and white-box testing. The row "risks tested" represents the number of risks initially selected for testing, as well as those not initially selected for testing (in parentheses). The row "vulnerabilities identified" represents the number of vulnerabilities identified by testing the risks initially selected for testing, as well as the number of vulnerabilities identified by testing the risks not initially selected (in parentheses). The four rows at the bottom of Table 1 provide statistics on the use of the various modeling constructs of CORAL to express the threat scenarios.

Table 1. Results obtained during black-box testing and white-box testing.

	Black-box	**White-box**	**Total**
SUT diagrams analyzed	11	2	13
Threat scenarios identified	27	4	31
Risks identified	19	2	21
Test cases identified	27	4	31
Risks testes	10 (plus 9)	1 (plus 1)	11 (plus 10)
Vulnerabilities identified	4 medium and 5 low (plus 0)	2 low (plus 2 low)	4 medium and 7 low (plus 2 low)
New messages	144	17	161
Altered messages	52	8	60
Deleted messages	10	7	17
Unwanted incidents	30	4	34

6 Discussion

The two variables that determine the risk level of a risk, that is, the frequency value and the consequence value, are estimates based on data gathered during the security risk assessment. In other words, these estimates tell us to what degree the identified risks exist. Thus, in principle, the higher the risk level, the more likely it is to reveal vulnerabilities causing the risk. The same applies the other way around. That is, the lower the risk level, the less likely it is to reveal vulnerabilities causing the risk.

The results obtained for RQ1 show that 11 vulnerabilities were revealed by testing the risks considered as most severe, while only 2 vulnerabilities were revealed by testing the risks considered as low risks. Additionally, the 2 vulnerabilities identified by testing the low risks were assigned low severity (see Table 1). These findings indicate that the risk levels of identified risks were quite accurate. In contrast, if we had found 2 vulnerabilities by testing the most severe risks, and 11 vulnerabilities by testing the low risks, then that would have indicated inaccurate risk levels, and thus a risk assessment of low quality. The results obtained for RQ2 show that we identified all relevant security risks compared to previous penetration tests. In addition, we identified five new security risks and did not leave out any risks of relevance for the features considered. In summary, the results obtained for RQ1 and RQ2 indicate that the produced risk models were valid and of high quality, and thus that the CORAL approach is effective in terms of producing valid risk models.

The results obtained for RQ3 point out that all threat scenarios were directly testable. We believe this result is generalizable because, in the CORAL approach, risks are identified at the level of abstraction testers commonly work when designing test cases [3]. This is also backed up by the fact that we made direct use of all threat scenarios as security test cases. Thus, the CORAL approach is effective in terms of producing threat scenarios that are directly testable. However, it is important to note that the CORAL approach is designed to be used by individual security testers, or by a group of security testers collaborating within the same testing project. The risk models produced by a tester, or a group of testers working together, will most likely be used by the same tester(s) to design test cases, and consequently execute the test cases.

In general, the CORAL approach seems to work equally well for black-box testing and white-box testing. Based on the results obtained for RQ4, we see that it is possible to carry out the complete CORAL approach both in black-box testing and white-box testing. The reason why Table 1 shows lower numbers in the white-box column compared to the numbers in the black-box column, is because we had fewer white-box models to analyze compared to black-box models.

Table 1 also shows that the threat scenarios mostly consisted of *new messages* and *altered messages*. Only 17 out of 272 messages were *deleted messages*. This may be an indication that threat scenarios can be sufficiently expressed without the usage of deleted messages. Nevertheless, they are important to document that an interaction is deleted by a threat. Note that the number of unwanted incidents (34) is greater than the number of identified risks (21). This is because some of the risks reoccurred in several threat scenarios, and thus had to be repeated in every threat scenario in which they reoccurred.

Another important observation is that the threat scenarios identified during white-box testing helped us locate where in the source code risks occurred, although the threat scenarios were initiated at the application level.

7 Related Work and Conclusions

In order to get a holistic picture, we relate the CORAL approach to other risk-driven testing approaches at a general level and not only to approaches focusing on security. In addition, because the CORAL approach is a model-based approach, we will relate our approach to other model-based approaches. The reader is referred to our systematic literature review for an overview of state of the art risk-driven testing approaches, where we also review approaches that are not model-based [4].

Most model-based approaches make use of fault tree analysis (attack tree is a variant of a fault tree) for the purpose of safety risk assessment [2, 8, 12, 15, 21, 27]. While fault tree analysis in these approaches are useful for identifying specific safety risks, they do not include information such as the threat profile initiating the chain of events causing the risks, the likelihood of an event occurring, and the consequence of risks. These constructs are necessary in a risk assessment [10]. Moreover, these approaches do not provide any guidelines for estimating or evaluating risks. The approaches leave it to the tester to identify the most severe risks in an ad hoc manner, and plan the testing process accordingly. In these approaches, test cases are designed by first analyzing the fault tree diagrams, and then modeling state machine diagrams (similar to UML state machines [16]) that represent test cases exploring the faults. However, there are some existing gaps between fault trees and state machines that need to be taken into consideration when modeling state machine diagrams based on fault tree diagrams [11]. In addition, while these approaches focus on modeling state-based test cases, our approach focuses on modeling interaction-based test cases.

The risk-driven security testing approaches provided by Botella et al. [1], Großmann et al. [9], and Seehusen [23] make use of the CORAS risk analysis language [14] for the purpose of security risk assessment. The graphical notation of the CORAL risk analysis language is based on the CORAS language. The CORAL approach is therefore closely related to these approaches. However, there are some fundamental differences. First, CORAS risk models represent threat scenarios and risks at a high-level of abstraction, while we represent these at a low-level of abstraction. Second, CORAS risk models are represented as directed acyclic graphs, while we represent risk models as sequence diagrams, which are better suited for model-based testing [3]. Third, these approaches use the risk estimates assigned on a CORAS risk model to make a prioritized list of threat scenarios which in turn represent a prioritized list of high-level test procedures [23]. The high-level test procedures are then used as a starting point for identifying/designing test cases either manually or by instantiating certain test patterns. In the CORAL approach, we map the risks to a risk evaluation matrix based on the risk estimates, and then we make a prioritized list of risks. We then select the most severe risks that the system under test is exposed to, and design test cases by making use of the CORAL risk models in which the selected risks occur.

The approach provided by Wendland et al. [25] makes use of behavior trees for the purpose of identifying risks at the level of requirements engineering. This approach does not identify risks by modeling threat scenarios, but rather by carrying out high-level qualitative risk assessment, and annotate the behavior trees with qualitative risk exposure values. The approach does not explicitly model test cases, but instead provides guidelines (referred to as test design strategy) testers may use to model test cases.

Zech et al. [26] identify security risks solely by matching attack patterns on the public interfaces of a system under test. The pattern matching is carried out automatically, which in turn produces risk models in terms of UML class diagrams [16]. The produced risk models contain information about possible attack scenarios that may be carried out on the public interfaces. However, the risk models do not contain information regarding the threat initiating the attacks, and the chain of events causing the security risks. The approach transforms the risk models into misuse case models, represented as class diagrams, from which test cases are generated.

What is common for all the approaches discussed above is that they model risks and the system under test in separate models using separate modeling languages. This makes it difficult to get an intuitive understanding with respect to exactly how and where the risks affect the system under test. The risk models in the CORAL approach represent specific threat scenarios, security risks caused by the threat scenarios, and the relevant aspects of the system affected by the risks, within the same model. This enables testers to identify exactly how and where certain security risks may occur.

In this report, we have presented an evaluation of CORAL based on our experiences from applying the approach in an industrial case study. The SUT in the case study was a web application designed to deliver streamlined administration and reporting of all forms of equity-based compensation plans. The objective of the case study was to evaluate to what extent the CORAL approach helps security testers in selecting and designing test cases. In the CORAL approach, we base the test selection and the test design on the risk models produced during the security risk assessment. Our hypothesis was that the produced risk models are valid, and that the threat scenarios represented by the risk models are directly testable.

The case study results indicate that the CORAL approach is effective in terms of producing valid risk models. This is backed up by two observations. First, we identified in total 21 risks, and 11 of these risks were considered as most severe, while the remaining 10 risks were considered as low risks. By testing these 11 risks we identified 11 vulnerabilities, while by testing the remaining

10 risks we identified only 2 vulnerabilities. Second, we identified all relevant security risks compared to previous penetration tests. In addition, we identified five new security risks and did not leave out any risks of relevance for the features considered.

The CORAL approach seems to work equally well for black-box and white-box testing. One point worth noting for white-box testing is that the threat scenarios help locating risks at the source code level although they are initiated at the application level.

Finally, one of the most important findings we did in the case study is that the CORAL approach is very useful for identifying security test cases. We used all threat scenarios identified in the case study for the purpose of security test case design and execution.

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) funded by the European Commission within the 7th Framework Programme.

References

1. J. Botella, B. Legeard, F. Peureux, and A. Vernotte. Risk-Based Vulnerability Testing Using Security Test Patterns. In *Proc. 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14)*, pages 337–352. Springer, 2014.
2. R. Casado, J. Tuya, and M. Younas. Testing long-lived web services transactions using a risk-based approach. In *Proc. 10th International Conference on Quality Software (QSIC'10)*, pages 337–340. IEEE Computer Society, 2010.
3. A.C. Dias Neto, R. Subramanyan, M. Vieira, and G.H. Travassos. A Survey on Model-based Testing Approaches: A Systematic Review. In *Proc. 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies (WEASELTech'07)*, pages 31–36. ACM, 2007.
4. G. Erdogan, Y. Li, R.K. Runde, F. Seehusen, and K. Stølen. 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, 2014.
5. G. Erdogan, A. Refsdal, and K. Stølen. A Systematic Method for Risk-driven Test Case Design Using Annotated Sequence Diagrams. In *Proc. 1st International Workshop on Risk Assessment and Risk-driven Testing (RISK'13)*, pages 93–108. Springer, 2014.
6. G. Erdogan, A. Refsdal, and K. Stølen. Schematic Generation of English-prose Semantics for a Risk Analysis Language Based on UML Interactions. In *Proc. 2nd International Workshop on Risk Assessment and Risk-driven Testing (RISK'14)*, pages 205–310. IEEE Computer Society, 2014.
7. Find Security Bugs V1.2.1. http://h3xstream.github.io/find-sec-bugs/. Accessed April 30, 2015.
8. M. Gleirscher. Hazard-based selection of test cases. In *Proc. 6th International Workshop on Automation of Software Test (AST'11)*, pages 64–70. ACM, 2011.

9. J. Großmann, M. Schneider, J. Viehmann, and M.-F. Wendland. Combining Risk Analysis and Security Testing. In *Proc. 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14)*, pages 322–336. Springer, 2014.
10. International Organization for Standardization. *ISO 31000:2009(E), Risk management – Principles and guidelines*, 2009.
11. H. Kim, W.E. Wong, V. Debroy, and D. Bae. Bridging the Gap Between Fault Trees and UML State Machine Diagrams for Safety Analysis. In *Proc. 17th Asia Pacific Software Engineering Conference (APSEC'10)*, pages 196–205. IEEE Computer Society, 2010.
12. J. Kloos, T. Hussain, and R. Eschbach. Risk-based testing of safety-critical embedded systems driven by Fault Tree Analysis. In *Proc. 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW'11)*, pages 26–33. IEEE Computer Society, 2011.
13. Lapse Plus Console V2.8.1. https://code.google.com/p/lapse-plus/. Accessed April 30, 2015.
14. M.S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer, 2011.
15. R. Nazier and T. Bauer. Automated risk-based testing by integrating safety analysis information into system behavior models. In *Proc. 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW'12)*, pages 213–218. IEEE Computer Society, 2012.
16. Object Management Group. *Unified Modeling Language (UML), superstructure, version 2.4.1*, 2011. OMG Document Number: formal/2011-08-06.
17. Object Management Group. *UML Testing Profile (UTP), version 1.2*, 2013. OMG Document Number: formal/2013-04-03.
18. Open Web Application Security Project. https://www.owasp.org/index.php/Main_Page. Accessed April 30, 2015.
19. OWASP Zed Attack Proxy. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. Accessed April 30, 2015.
20. B. Potter and G. McGraw. Software Security Testing. *Security & Privacy, IEEE*, 2(5):81–85, 2004.
21. M. Ray and D.P. Mohapatra. Risk analysis: A guiding force in the improvement of testing. *IET Software*, 7:29–46, 2013.
22. P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.
23. F. Seehusen. A Technique for Risk-Based Test Procedure Identification, Prioritization and Selection. In *Proc. 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14)*, pages 277–291. Springer, 2014.
24. Visual Code Grepper V2.0.0. http://sourceforge.net/projects/visualcodegrepp/. Accessed April 30, 2015.
25. M.-F. Wendland, M. Kranz, and I. Schieferdecker. A systematic approach to risk-based testing using risk-annotated requirements models. In *Proc. 7th International Conference on Software Engineering Advances (ICSEA'12)*, pages 636–642. IARA, 2012.
26. P. Zech, M. Felderer, and R. Breu. Towards a Model Based Security Testing Approach of Cloud Computing Environments. In *Proc. 6th International Conference on Software Security and Reliability Companion (SERE-C'12)*, pages 47–56. IEEE Computer Society, 2012.

27. F. Zimmermann, R. Eschbach, J. Kloos, and T. Bauer. Risk-based statistical testing: A refinement-based appraoch to the reliability analysis of safety-critical systems. In *Proc. 12th European Workshop on Dependable Computing (EWDC'09)*, pages 1–8. The Open Archive HAL, 2009.

Technology for a better society

www.sintef.no