

A Systematic Method for Risk-Driven Test Case Design Using Annotated Sequence Diagrams

Gencer Erdogan^{1,2}(✉), Atle Refsdal¹, and Ketil Stølen^{1,2}

¹ Department for Networked Systems and Services, SINTEF ICT,
P.O. Box 124 Blindern, 0314 Oslo, Norway

² Department of Informatics, University of Oslo,
P.O. Box 1080 Blindern, 0316 Oslo, Norway

{gencer.erdogan,atle.refsdal,ketil.stolen}@sintef.no

Abstract. Risk-driven testing is a testing approach that aims at focusing the testing process on the aspects or features of the system under test that are most exposed to risk. Current risk-driven testing approaches succeed in identifying the aspects or features that are most exposed to risks, and thereby support testers in planning the testing process accordingly. However, they fail in supporting testers to employ risk analysis to systematically design test cases. Because of this, there exists a gap between risks, which are often described and understood at a high level of abstraction, and test cases, which are often defined at a low level of abstraction. In this paper, we bridge this gap. We give an example-driven presentation of a novel method, intended to assist testers, for systematically designing test cases by making use of risk analysis.

1 Introduction

Risk-driven testing (or risk-based testing) is a testing approach that use risk analysis within the testing process [4]. The aim in risk-driven testing is to focus the testing process with respect to certain risks of the system under test (SUT).

However, current risk-driven testing approaches leave a gap between risks, which are often described and understood at a high level of abstraction, and test cases, which are often defined at a low level of abstraction. The gap exists because risk analysis, within risk-driven testing approaches, is traditionally used as a basis for planning the test process rather than designing the test cases. Making use of risk analysis when planning the test process helps the tester to focus on the systems, aspects, features, use-cases, etc. that are most exposed to risk, but it does not support test case design. In order to bridge the gap between risks and test cases, risk-driven testing approaches should not merely make use of the risk analysis when planning the test process, but also when designing test cases. Specifically, risk-driven testing approaches must provide testers with steps needed to design test cases by making use of the risk analysis.

In this paper, we present a systematic and general method, intended to assist testers, for designing test cases by making use of risk analysis. A test case is a behavioral feature or behavior specifying tests [15]. We employ UML sequence diagrams [14] as the modeling language, conservatively extended with our own notation for representing risk information. In addition, we make use of the UML Testing Profile [15] to specify test cases in sequence diagrams. The reason for choosing sequence diagrams is that they are widely recognized and used within the testing community. In fact, it is among the top three modeling languages applied within the model-based testing community [13]. By annotating sequence diagrams with risk information, we bring risk analysis to the work bench of testers without the burden of a separate risk analysis language, thus reducing the effort needed to adopt the approach. Recent surveys on trends within software testing show that the lack of time and high costs are still the dominating barriers to a successful adoption of testing methods and testing tools within IT organizations [5].

Our method consists of four steps. In Step 1, we analyze the SUT and identify threat scenarios and unwanted incidents with respect to the relevant assets. In Step 2, we estimate the likelihood of threat scenarios and unwanted incidents, as well as the consequence of unwanted incidents. In Step 3, we prioritize risks, and then for each risk we prioritize paths that lead to the risk. In Step 4, we design test cases with respect to the paths selected for testing.

Section 2 gives an overview of our method. Section 3 introduces the web application on which we apply our method to demonstrate its applicability. Sections 4–7 employ the four steps on the web application, respectively. Section 8 relates our method to current risk-driven testing approaches that also address test case design. Finally, we provide concluding remarks in Sect. 9.

2 Overview of Method

Before going into the details of our method, we explain the assumed context in which it is to be applied. A testing process starts with *test planning*, followed by *test design and implementation*, *test environment set-up and maintenance*, *test execution*, and finally *test incident reporting* [9]. Our method starts after test planning, but before test design and implementation. Furthermore, the first and the fourth step in our method expect as input a description of the SUT in terms of sequence diagrams and suspension criteria, respectively. Suspension criteria are criteria used to stop all or a portion of the testing activities [8]. This is also known as test stopping criteria or exit criteria. Suspension criteria are used in our method to reflect the investable testing effort. We assume that these inputs are obtained during test planning. Next, we assume that the preparations for carrying out risk analysis have been completed, i.e., that assets have been identified, likelihood and consequence scales have been defined, and a risk evaluation matrix has been prepared so that risks can later be inserted as soon as their likelihood and consequence are determined. Our method consists of four main steps as illustrated in Fig. 1; dashed document icons represent input prepared

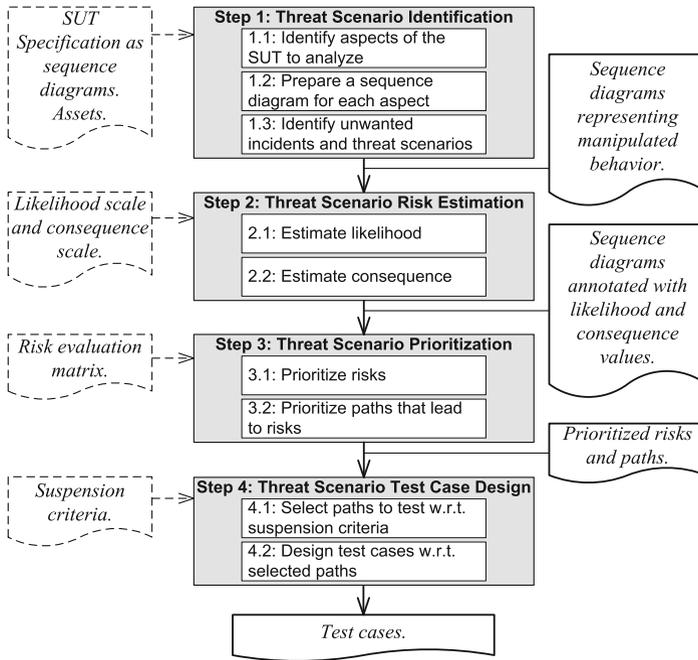


Fig. 1. Overview of the steps in the method.

during test planning, solid document icons represent output from one step and acts as input to the following step.

In **Step 1**, we analyze the SUT with the objective to identify unwanted incidents with respect to a certain asset to be protected, as well as threat scenarios resulting from manipulations initiated by the threat. This step expects as input a sequence diagram specification of the SUT and the asset that is to be considered. **First**, we identify the aspects of the SUT we are interested in analyzing. We then annotate each aspect with a label, containing a unique identifier. **Second**, we prepare a corresponding sequence diagram to capture risk information for each aspect label. Each sequence diagram inherits the SUT specification encapsulated by the underlying aspect label. Additionally, it represents the asset as a lifeline. The threats that may initiate threat scenarios are also represented as lifelines. **Third**, we identify unwanted incidents that have an impact on the asset, and threat scenarios that may lead to the unwanted incidents. The output of this step is a set of annotated sequence diagrams that represent manipulated behavior of the SUT and its context, in terms of threat scenarios and unwanted incidents.

In **Step 2**, we estimate the likelihood for the occurrence of the threat scenarios and the unwanted incidents in terms of frequencies, the conditional probability for threat scenarios leading to other threat scenarios or to unwanted incidents, as well as the impact of unwanted incidents on the asset. The input for this step is the output of Step 1. Additionally, this step expects a predefined

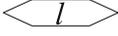
likelihood scale in terms of frequencies, and a predefined consequence scale in terms of impact on the asset. **First**, we estimate the likelihood for the occurrence of the threat scenarios and the unwanted incidents using the likelihood scale, as well as the conditional probability for threat scenarios leading to other threat scenarios or to unwanted incidents. **Second**, we estimate the consequence of unwanted incidents using the consequence scale. The output of this step is the same set of sequence diagrams given as the input for the step, annotated with likelihood estimates and consequence estimates as described above. A risk in our method is represented by an unwanted incident (i.e., a message to the asset lifeline) together with its likelihood value and its consequence value. Different sets of threat scenarios may lead to the same risk. We refer to the different sets of threat scenarios as different *paths*. That is, different paths may lead to the same risk.

In **Step 3**, we prioritize the risks as well as the different paths that lead to the prioritized risks. The input for this step is the output of Step 2. Additionally, this step employs the predefined risk evaluation matrix. **First**, we map all the risks to the risk evaluation matrix based on their likelihood (frequency) value and consequence (impact) value. We then prioritize the risks based on their risk level, i.e., their position in the risk evaluation matrix. **Second**, we prioritize the different paths that lead to the selected risks with respect to the likelihood-contribution of each path. The output of this step is a prioritized list of risks, and a prioritized list of paths leading to the prioritized risks.

In **Step 4**, we design test cases with respect to paths leading to selected risks. The input for this step is the output of Step 1 and the output of Step 3. Additionally, this step expects predefined suspension criteria. **First**, we select risks we would like to test based on the prioritized list of risks, and then we select paths, based on the prioritized list of paths, leading up to the selected risks. These selections are done with respect to the predefined suspension criteria. **Second**, for each path we want to test, we refer to its sequence diagram identified in Step 1 and use that as a basis for specifying a test case. We specify a test case by annotating the sequence diagrams using the UML Testing Profile [15]. The output of this step is a set of sequence diagrams representing test cases.

Table 1 shows the notation for annotating sequence diagrams with risk information. We have mapped some risk information to corresponding UML constructs for sequence diagrams. Assets and threats are represented as lifelines. Inspired by CORAS [11], we distinguish between three types of threats; deliberate threats (the leftmost lifeline in the Notation column), unintentional threats (the center lifeline in the Notation column) and non-human threats (the rightmost lifeline in the Notation column). Manipulations and unwanted incidents are represented as messages. We distinguish between three types of manipulations; new messages in the sequence diagram (a message annotated with a filled triangle), alteration of existing messages in the sequence diagram (a message annotated with an unfilled triangle), and deletion of existing messages in the sequence diagram (a message annotated with a cross inside a triangle). Aspect labels, likelihoods, conditional probabilities and consequences do not have

Table 1. Notation for annotating sequence diagrams with risk information.

Risk information	UML Construct	Notation
Aspect label	N/A	
Asset	Lifeline	
Threat	Lifeline	
Manipulation	Message	
Unwanted incident	Message	
Likelihood	N/A	
Conditional probability	N/A	
Consequence	N/A	

corresponding UML constructs for sequence diagrams. However, the following constraints apply: A likelihood can only be attached horizontally across lifelines. A likelihood assignment represents the likelihood, in terms of frequency, of the interaction preceding the likelihood assignment. The purpose of messages representing unwanted incidents is to denote that an unwanted incident has an impact on an asset. A consequence can therefore only be attached on messages representing unwanted incidents. A conditional probability may be attached on any kind of message except messages representing unwanted incidents. A conditional probability assignment represents the probability of the occurrence of the message on which it is assigned, given that the interaction preceding the message has occurred.

3 Example: Guest Book Application

As mentioned in Sect. 1, our method is a general method for designing test cases by making use of risk analysis. In this demonstration, we focus on security, and apply the steps presented in Sect. 2 on a guest book that is available in the Damn Vulnerable Web Application (DVWA) [3]. One of DVWA's main goals is to be an aid for security professionals to test their skills and tools in a legal environment [3]. DVWA is programmed in the scripting language PHP and requires a dedicated MySQL server to function correctly. We are running DVWA version 1.8 on the HTTP server XAMPP version 1.8.2 [22], which provides the required execution environment.

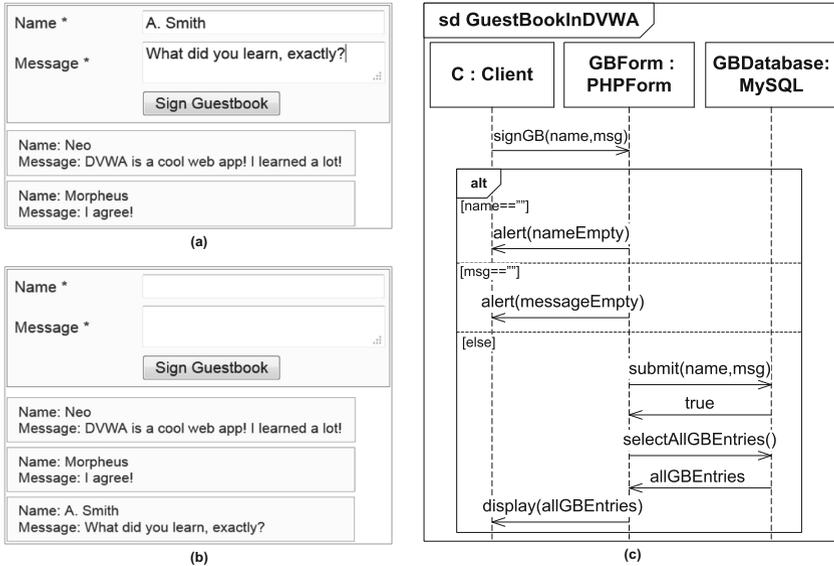


Fig. 2. (a) Screenshot of the guest book before submitting a new entry. (b) Screenshot of the guest book after submitting the entry. (c) Specification of the guest book expressed as a sequence diagram.

The SUT in this demonstration is a guest book in DVWA. Figure 2a shows a screenshot of the guest book user interface before a guest book entry is submitted, while Fig. 2b shows a screenshot of the user interface after the guest book entry is successfully submitted. Figure 2c represents its behavioral specification expressed as a sequence diagram. A guest book user may use a web browser in a client to sign the guest book by typing a name and a message, and then submit the guest book entry by clicking the “Sign Guestbook” button. If the *name* input field is empty, the guest book form replies with a warning message. If the *name* input field is not empty, but the *message* input field is empty, the guest book form also replies with a warning message. If neither of the input fields are empty, the guest book form submits the entry to the guest book database. The guest book database stores the entry and replies with the message *true* indicating that the transaction was successful. Having received the message *true*, the guest book form retrieves all of the guest book entries from the database, including the one just submitted, and displays them to the client.

4 Step 1: Threat Scenario Identification

The SUT in this demonstration is the guest book explained in Sect. 3. Let us assume that we are interested in analyzing the guest book with respect to a security asset defined as “integrity of guest-book’s source code”.

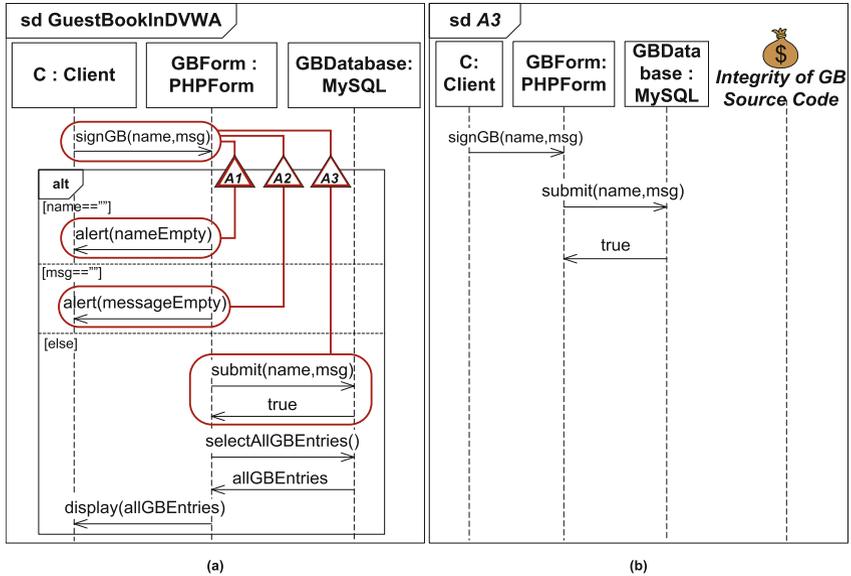


Fig. 3. (a) Specification of the guest book annotated with aspect labels. (b) Corresponding sequence diagram of the aspect encapsulated by aspect label A3.

As shown in Fig. 3a, we have identified three aspects labeled with aspect labels A1, A2 and A3. For the aspect represented by aspect label A1, we are interested in analyzing the interaction composed of the messages `signGB(name,msg)` and `alert(nameEmpty)`, with respect to the integrity of the guest-book's source code. The same reasoning applies for A2 and A3. The aspects identified in this example are small. In practice it may well be that one is interested in analyzing bigger and more complex aspects. The granularity level of the aspects is determined by the tester.

Suppose we are only interested in analyzing the aspect encapsulated by aspect label A3. Figure 3b shows a sequence diagram corresponding to the interaction encapsulated by aspect label A3. Additionally, it represents the abovementioned security asset as a lifeline. We now have a sequence diagram we can use as a starting point to analyze the SUT aspect encapsulated by aspect label A3, with respect to integrity of the guest-book's source code. We represent the risk related information in bold and italic font, in the sequence diagrams, to distinguish between the specification and the risk related information.

We proceed the analysis by identifying unwanted incidents that may have an impact on the security asset, and threat scenarios that may lead to the unwanted incidents (see Fig. 4). The integrity of the guest-book's source code is compromised if, for example, a malicious script is successfully stored (i.e., injected) in the guest book database. A malicious script that is injected in the guest book database is executed by the web browser of the guest book user when accessed. This modifies the content of the HTML page on the user's web browser, thus

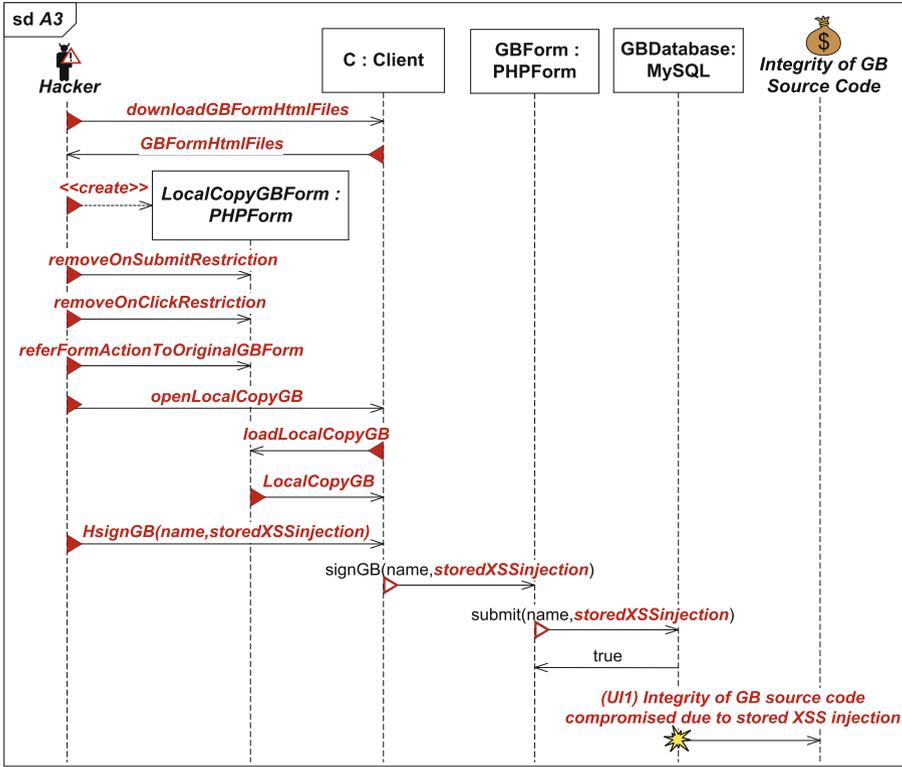


Fig. 4. Identifying unwanted incidents and threat scenarios for the aspect encapsulated by aspect label A3, w.r.t. integrity of the guest-book’s source code.

compromising the integrity of the guest-book’s source code. These kinds of script injections are also known as stored cross-site scripting (stored XSS) [17]. We identify this as an unwanted incident (UI1), as represented by the last message in Fig. 4.

UI1 may be caused by different manipulations on the expected behavior of the guest book. One potential cause could be that the *msg* parameter in *submit(name, msg)* and *signGB(name,msg)* is replaced with *storedXSSinjection*, representing an XSS injection script. This is an alteration of the guest-book’s expected behavior. We therefore replace the messages *signGB(name,msg)* and *submit(name,msg)* with messages representing alterations.

These alterations may be initiated by different threats. Let us say we are interested in analyzing this further from a hacker perspective, which is categorized as a deliberate threat. A hacker may successfully carry out an XSS injection by, for example, first downloading the HTML files of the guest book using the web browser, in order to create a local copy of the guest-book’s user interface (*downloadGBFormHtmlFiles*, *GBFormHtmlFiles* and *<<create>>*). Having successfully saved a local copy of the guest-book’s HTML files, the hacker removes all

restrictions, such as the maximum number of characters allowed in the name and message input fields when submitting a guest book entry (*removeOnSubmitRestriction* and *removeOnClickRestriction*). Then, the hacker refers all actions to the original guest book by making use of its web address (*referFormActionToOriginalGBForm*). Finally, the hacker loads the local copy of the guest book in the web browser, writes an XSS injection script in the message field, and submits the guest book entry containing the XSS injection (*openLocalCopyGB*, *loadLocalCopyGB*, *LocalCopyGB* and *HsignGB(name,storedXSSinjection)*). Note that all of the messages described in this paragraph are annotated as new messages in the sequence diagram (message with a filled triangle).

5 Step 2: Threat Scenario Risk Estimation

Table 2 shows the likelihood and consequence scale that we assume have been established during preparation of the risk analysis. The likelihood scale is given in terms of frequency intervals. The description of likelihood Rare reads “zero to ten times per year”. The description of likelihoods Unlikely, Possible and Likely reads in similar way, while the description for likelihood Certain reads “three hundred times or more per year”. The consequence scale is given in terms of impact on different categories of the security asset. For example, an unwanted incident has a catastrophic impact on the security asset if it compromises the integrity of the guest-book’s source code that carries out database transactions. Similar interpretations apply for the other consequences.

Figure 5 shows likelihood estimates for the threat scenarios and the unwanted incident identified in Step 1, as well as a consequence estimate for the unwanted incident. The tester may estimate likelihood values and consequence values based on expert judgment, statistical data, a combination of both, etc. Let us say we have acquired information indicating that hackers most likely prepare injection attacks in the manner described by the interaction starting with message *downloadGBFormHtmlFiles*, and ending with message *LocalCopyGB* in Fig. 5. For this reason, we choose to assign likelihood Likely on this interaction. Note that Likely corresponds to the frequency interval $[150, 300>:1y$ (see Table 2).

XSS injection attacks are less likely to be initiated by hackers compared to other kinds of injection attacks they initiate (such as SQL-injection) [18].

Table 2. Likelihood scale and consequence scale.

Likelihood scale		Consequence scale	
Likelihood	Description	Consequence	Description
Rare	$[0, 10>:1y$	Insignificant	Src. that generates the aesthetics.
Unlikely	$[10, 50>:1y$	Minor	Src. that retrieves third party ads.
Possible	$[50, 150>:1y$	Moderate	Src. that generates the user interface.
Likely	$[150, 300>:1y$	Major	Src. that manages sessions and cookies.
Certain	$[300, \dots>:1y$	Catastrophic	Src. that carries out database transactions.

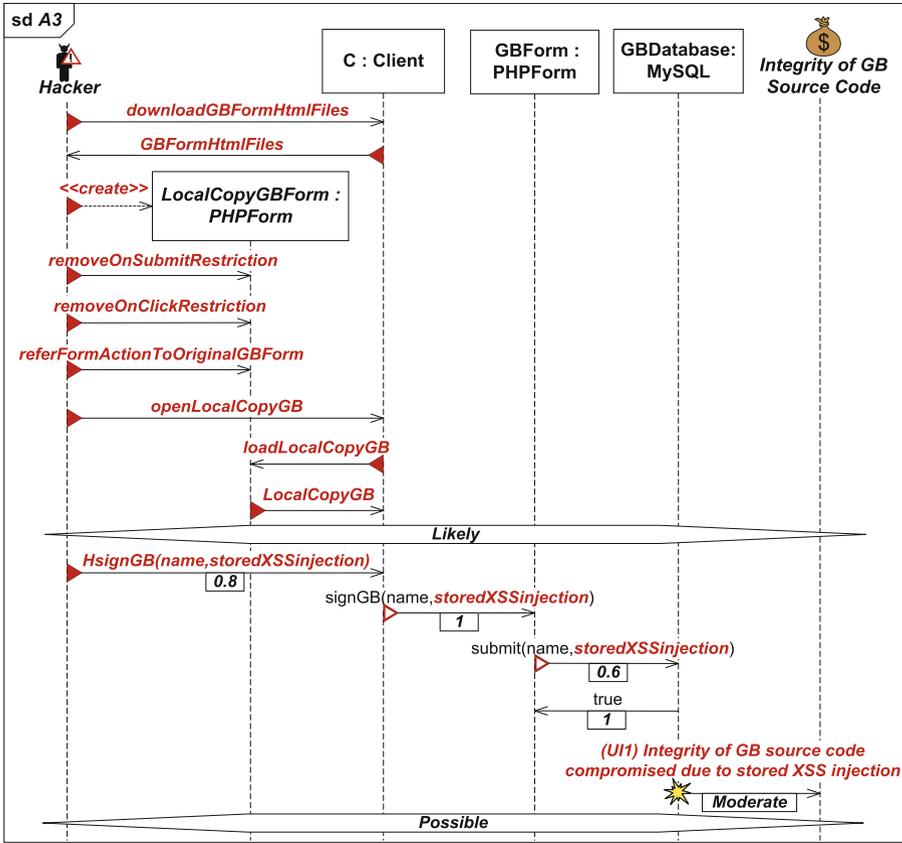


Fig. 5. Estimating the likelihood of the threat scenarios and unwanted incident *UI1*, as well as the consequence of *UI1*.

For this reason, we choose to assign a probability 0.8 on message $HsignGB(name, storedXSSinjection)$, indicating that it will occur with probability 0.8 given that the messages preceding it has occurred. This probability assignment leads to a different frequency interval for the interaction starting with message $downloadGBFormHtmlFiles$ and ending with message $HsignGB(name, storedXSSinjection)$. The frequency interval for the aforementioned interaction is calculated by multiplying $[150, 300] > :1y$ with 0.8 , which results in the frequency interval $[120, 240] > :1y$. This frequency interval is in turn used to calculate the subsequent frequency interval, in the path, in a similar manner. This procedure is carried out until the frequency interval for the whole path leading to the unwanted incident is calculated. The frequency interval for the whole path is then mapped to the likelihood scale in Table 2 in order to deduce a likelihood value. The deduced likelihood value represents the likelihood value for the whole path, and thereby the likelihood value for the unwanted incident.

We proceed the estimation by identifying conditional probabilities for the remaining messages. We assume message $signGB(name,storedXSSinjection)$ will occur with probability 1 since the hacker has removed all restrictions on the local copy of the guest book form. The guest book form is programmed in the scripting language PHP. Although PHP makes use of what is known as “prepared statements” to validate input directed to the database, bypassing the validation is still possible if the prepared statements are not handled correctly [19]. These kinds of bypasses require insight into the structure of the source code and are therefore harder to exploit. For this reason, we choose to assign a probability 0.6 on message $submit(name,storedXSSinjection)$. We assume message $true$ will occur with probability 1, as there is nothing that prevents the database from executing the query containing the XSS injection if it has made all its way into the database.

We calculate the frequency interval for the whole path by multiplying $[150, 300>:1y$ with the product of the abovementioned conditional probabilities. That is, we multiply $[150, 300>:1y$ with 0.48, which results in the frequency interval $[72, 144>:1y$. By mapping this frequency interval to the likelihood scale in Table 2, we see that the frequency interval is within the boundaries of likelihood Possible. This means that unwanted incident $UI1$ may occur with likelihood Possible. Finally, a stored XSS injection has the objective to execute a script on the end user’s web browser for different purposes. This means that stored XSS injection modifies the source code that generates the user interface. Thus, $UI1$ has an impact on the security asset with a moderate consequence.

6 Step 3: Threat Scenario Prioritization

Table 3 shows the risk evaluation matrix established during preparation of the risk analysis. In traditional risk analysis, risk evaluation matrices are designed to group the various combinations of likelihood and consequence into three to five risk levels (e.g., low, medium and high). Such risk levels cover a wide spectrum of likelihood and consequence combinations and are typically used as a basis for deciding whether to accept, monitor or treat risks. However, in risk-driven testing setting, one is concerned about prioritizing risks to test certain aspects of the SUT exposed to risks. A higher granularity with respect to risk levels may therefore be more practical. The risk evaluation matrix in Table 3 represents nine risk levels, diagonally on the matrix. The tester defines the interpretation of the risk levels. In this demonstration we let numerical values represent risk levels; [1] represents the lowest risk level and [9] represents the highest risk level.

In Step 2, we estimated that $UI1$ occurs with likelihood Possible and has a moderate impact on the security asset. Based on these estimations, we map $UI1$ to its respective cell in the risk evaluation matrix. In order to demonstrate the prioritizing of risks for testing, we assume that we have identified two additional risks $UI2$ (with likelihood Likely and consequence Insignificant) and $UI3$ (with likelihood Unlikely and consequence Moderate) in the same manner as we identified $UI1$. We map $UI2$ and $UI3$ to the risk evaluation matrix with respect

Table 3. Risk evaluation matrix composed of the scales in Table 2.

		Consequence				
		Insignificant	Minor	Moderate	Major	Catastrophic
Likelihood	Rare	[1]				
	Unlikely	[2]		<i>UI3</i>		
	Possible	[3]		<i>UI1</i>		
	Likely	[4] <i>UI2</i>				
	Certain	[5]	[6]	[7]	[8]	[9]

to their likelihood value and consequence value. The result is shown in Table 3. We prioritize the risks according to their risk level. *UI1*, *UI2* and *UI3* has risk level [5], [4], and [4], respectively. Note that *UI2* and *UI3* have the same risk level. It is the task of the tester to prioritize among such risks, if necessary, and this must always be justified. This concludes the prioritization of risks. Next, we prioritize among different paths that lead to the same risk.

As mentioned in Sect. 2, different sets of threat scenarios, i.e., different paths, may lead to the same risk. Figure 5 shows the estimation of one path that leads to *UI1*. Let us name this path *P1*. Suppose we have identified a second path *P2* that also leads to *UI1*. In *P2*, XSS injection is carried out via a man-in-the-middle attack on the HTTPS connection between the client and the guest book. Suppose we have carried out an estimation for *P2* and arrived at likelihood Rare because man-in-the-middle attacks on the HTTPS connection are unlikely to be successful, due to guest-book’s usage of proper countermeasures, e.g., as presented in [16]. *P1* and *P2* represent two separate paths that lead to *UI1*. This means that the frequency for *UI1* is the sum of the frequency interval in *P1* and the frequency interval in *P2*, which results in the new frequency interval $[50, 160]:1y$. By mapping the new frequency interval to the likelihood scale in Table 2, we see that it overlaps the likelihoods Possible and Likely. However, the frequency interval is skewed more towards Possible than Likely. We therefore choose to keep *UI1* within likelihood Possible. We prioritize the paths according to their likelihood contribution. *P1* has a higher likelihood contribution (Possible) on *UI1* than *P2* (Rare). This concludes the prioritization of paths.

7 Step 4: Threat Scenario Test Case Design

Suppose, for the sake of the example, the following suspension criteria is given: “Only test paths with likelihood contribution Possible or higher on risks within risk level [5]”. We see from the prioritization in Step 3 that *UI1* has risk level [5], and that *P1* has likelihood contribution Possible on *UI1*, while *P2* has likelihood contribution Rare on *UI1*. We select *P1* for testing.

We specify our test cases by annotating sequence diagrams using the stereotypes given in the UML Testing Profile [15]: The stereotype <<SUT>> is applied to one or more properties of a classifier to specify that they constitute the system under test. The stereotype <<TestComponent>> is used to

represent a component that is a part of the test environment which communicates with the SUT or other test components. Test components are used in test cases for stimulating the SUT with test data and for evaluating whether the responses of the SUT adhere with the expected ones. The stereotype <<ValidationAction>> is used on execution specifications, on lifelines representing test components, to set verdicts in test cases. The UML Testing Profile defines the following five verdicts: None (the test case has not been executed yet), pass (the SUT adheres to the expectations), inconclusive (the evaluation cannot be evaluated to be pass or fail), fail (the SUT differs from the expectation) and error (an error has occurred within the testing environment). The number of verdicts may be extended, if required.

A path represents a manipulated behavior of the system under test that leads to an unwanted incident. We design test cases with respect to paths. The test objective, for a test case designed with respect to a path, is to verify the validity of the manipulations posed on the system under test. In *P1* the integrity of guest-book's source code is compromised due to an XSS injection on the guest book database via the guest book form. The test objective for the test case designed with respect to *P1* is to verify the validity of these manipulations, i.e., to verify whether it is possible to successfully carry out the manipulations.

Figure 6 shows the test case designed with respect to *P1*. We annotate lifelines GBForm and GBDatabase with stereotype <<SUT>> to specify that they constitute the system under test. We annotate lifeline C with stereotype <<TestComponent>> to specify that it is a part of the test environment which communicates with the SUT. We add an execution specification on lifeline C annotated with stereotype <<ValidationAction>> to set the verdict for the test case. The verdict is set to fail meaning that the SUT differs from the expected behavior.

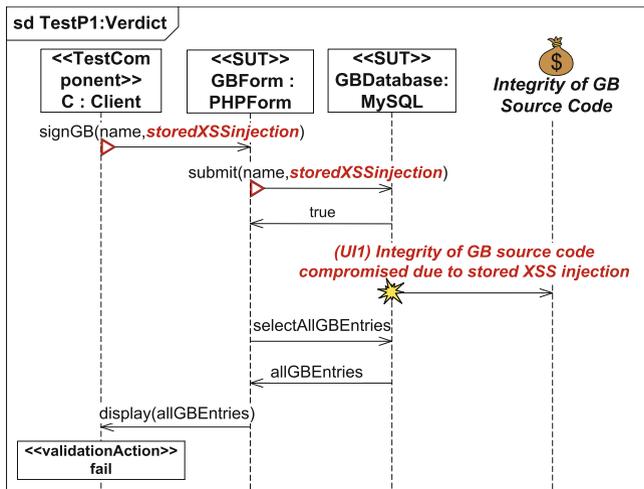


Fig. 6. Security test case designed w.r.t. the path represented in Fig. 4.

In other words, if stored XSS injection is successfully carried out then the SUT differs from the expected behavior, which should be to prevent XSS injections.

8 Related Work

Although risk analysis, within risk-driven testing, is traditionally used as a basis for planning the test process, few approaches also provide guidelines for deriving test cases as part of the approach. These approaches explain the process of identifying, estimating and prioritizing risks either partly or by briefly mentioning it. In [1, 10], risks are identified by making use of fault tree analysis, however, there is no explanation on how to estimate and prioritize the risks. In [6], the authors refer to fault tree analysis for identifying risks. There is no explanation on how to estimate and prioritize risks. In [12], the authors refer to a risk analysis approach published by NIST [21] for identifying security risks. However, there is no further explanation on how to identify and estimate the security risks, yet, security risks are prioritized with respect to a predefined risk assessment matrix. In [23], security risks are identified solely by matching attack patterns on the public interfaces of a SUT. The estimation and prioritization of risks are only based on a complexity factor for specific operations in the SUT. In practice, other factors may be considered, e.g., vulnerability statistics and incident reports. In [2], test cases are prioritized by calculating a risk exposure for test cases, with the objective to quantitatively measure the quality of test cases. Risk estimation is carried out by multiplying the probability of a fault occurring with the costs related to the fault. However, there is no explanation about how risks are identified. In [20], risks are estimated by multiplying the probability that an entity contains fault with the associated damage. Similar to [2], this value is used to prioritize test cases, and there is no explanation about how risks are identified.

All of these approaches use separate modeling languages or techniques for representing the risk analysis and the test cases: In [1, 6, 10], fault trees are used to identify risks, while test cases are derived from state machine diagrams with respect to information provided by the fault trees. In [12], high level risks are detailed by making use of threat modeling. Misuse cases are developed with respect to the threat models, which are then used as a basis for deriving test cases represented textually. In [23], risk models are generated automatically by making use of a vulnerability knowledge database. The risk models are used as input for generating misuse cases, which are also identified in similar manner. Misuse cases are used as a basis for deriving test cases. In [2, 20], a test case is a path in an activity diagram, starting from the activity diagram's initial node and ending at its final node. In [2], risks are estimated using tables, while in [20], risk information is annotated on the activities of an activity diagram, only in terms of probability, damage and their product.

9 Conclusion

In order to bridge the gap between high level risks and low level test cases, risk-driven testing approaches must provide testers with a systematic method for designing test cases by making use of the risk analysis. Our method is specifically designed to meet this goal.

The method starts after test planning, but before test design, according to the testing process presented by ISO/IEC/IEEE 29119 [9]. It brings risk analysis to the work bench of testers because it employs UML sequence diagrams as the modeling language, conservatively extended with our own notation for representing risk information. Sequence diagrams are widely recognized and used within the testing community and it is among the top three modeling languages applied within the model based testing community [13]. Risk identification, estimation and prioritization in our method are in line with what is referred to as risk assessment in ISO 31000 [7]. Finally, our approach makes use of the UML Testing Profile [15] to specify test cases in sequence diagrams. This means that our method is based on widely accepted standards and languages, thus facilitating adoption among the software testing community.

Acknowledgments. This work has been conducted as a part of the DIAMONDS project (201579/S10) funded by the Research Council of Norway, the NessoS network of excellence (256980) and the RASEN project (316853) funded by the European Commission within the 7th Framework Programme, as well as the CONCERTO project funded by the ARTEMIS Joint Undertaking (333053) and the Research Council of Norway (232059).

References

1. Casado, R., Tuya, J., Younas, M.: Testing long-lived web services transactions using a risk-based approach. In: Proceedings of 10th International Conference on Quality Software (QSIC'10), pp. 337–340. IEEE Computer Society (2010)
2. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proceedings of 2002 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'02), pp. 1–14. IBM Press (2002)
3. Damn Vulnerable Web Application (DVWA). <http://www.dvwa.co.uk/>. Accessed 11 Aug 2013
4. Erdogan, G., Li, Y., Runde, R.K., Seehusen, F., Stølen, K.: Conceptual framework for the DIAMONDS project. Technical report A22798, SINTEF Information and Communication Technology (2012)
5. Garousi, V., Zhi, J.: A survey of software testing practices in Canada. *J. Syst. Softw.* **86**(5), 1354–1376 (2013)
6. Gleirscher, M.: Hazard-based selection of test cases. In: Proceedings of 6th International Workshop on Automation of Software Test (AST'11), pp. 64–70. ACM (2011)
7. International Organization for Standardization. ISO 31000:2009(E), Risk management - Principles and guidelines (2009)

8. International Organization for Standardization. ISO/IEC/IEEE 29119-1:2013(E), Software and system engineering - Software testing - Part 1: Concepts and definitions (2013)
9. International Organization for Standardization. ISO/IEC/IEEE 29119-2:2013(E), Software and system engineering - Software testing - Part 2: Test process (2013)
10. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: Proceedings of 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW'11), pp. 26-33. IEEE Computer Society (2011)
11. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach. Springer, Heidelberg (2011)
12. Murthy, K.K., Thakkar, K.R., Laxminarayan, S.: Leveraging risk based testing in enterprise systems security validation. In: Proceedings of 1st International Conference on Emerging Network Intelligence (EMERGING'09), pp. 111-116. IEEE Computer Society (2009)
13. Dias-Neto, A.C., Subramanyan, R., Vieira, M., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. In: Proceedings of 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies (WEASELTech'07), pp. 31-36. ACM (2007)
14. Object Management Group. Unified Modeling Language (UML), superstructure, version 2.4.1, 2011. OMG Document Number: formal/2011-08-06
15. Object Management Group. UML Testing Profile (UTP), version 1.2, 2013. OMG Document Number: formal/2013-04-03
16. Oppliger, R., Hauser, R., Basin, D.: SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle. *Comput. Commun.* **29**(12), 2238-2246 (2006)
17. Open Web Application Security Project (OWASP). [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)). Accessed 5 Sept 2013
18. OWASP Top 10 Application Security Risks - 2013. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Accessed 6 Sept 2013
19. PHP manual. <http://php.net/manual/en/pdo.prepared-statements.php>. Accessed 6 Sept 2013
20. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of 3rd International Workshop on Automation of Software Test (AST'08), pp. 67-70. ACM (2008)
21. Stoneburner, G., Goguen, A., Feringa, A.: Risk Management Guide for Information Technology Systems. NIST Special Publication 800-30, National Institute of Standards and Technology (2002)
22. XAMPP. <http://www.apachefriends.org/en/xampp.html>. Accessed 11 Aug 2013
23. Zech, P., Felderer, M., Breu, R.: Towards a model based security testing approach of cloud computing environments. In: Proceedings of 6th International Conference on Software Security and Reliability Companion (SERE-C'12), pp. 47-56. IEEE Computer Society (2012)