SINTEF A26036 - Unrestricted

Report

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

Author(s)
Gencer Erdogan, Atle Refsdal, and Ketil Stølen

SINTEF ICT
Networked Systems and Services
2014-03-24

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:
Risk-driven testing,
Risk-based testing,
Testing,
Risk analysis,
Test case,
Test case design,
Security testing,
Security test case

Report

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

VERSION
Final

DATE
2014-03-24

AUTHOR(S)
Gencer Erdogan, Atle Refsdal, and Ketil Stølen

CLIENT(S)
Norwegian Research Council

CLIENT'S REF.
201579/S10

PROJECT NO.
102002253

NUMBER OF PAGES/APPENDICES:
36/0

ABSTRACT
Risk-driven testing is a testing approach that aims at focusing the testing 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 report, 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.

PREPARED BY Gencer Erdogan	SIGNATURE Gencer Erdogan
CHECKED BY Fredrik Seehusen	SIGNATURE [illegible]
APPROVED BY Bjørn Skjellaug	SIGNATURE [illegible]

REPORT NO.	**ISBN**	**CLASSIFICATION**	**CLASSIFICATION THIS PAGE**
SINTEF A26036	978-82-14-05349-4	Unrestricted	Unrestricted

Table of Contents

1 Introduction

Risk-driven testing (or risk-based testing) is a testing approach that use risk analysis within the testing process [5]. 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 report, 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 [16]. We employ UML sequence diagrams [15] as the modeling language, conservatively extended with our own notation for representing risk information. In addition, we make use of the UML Testing Profile [16] 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 [14]. 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 [6].

Our method consists of four steps. In Step 1, we analyze the SUT and identify threat scenarios and unwanted incidents with respect to 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 and select paths which consist of sequences of threat scenarios leading up to and including a 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, 5, 6 and 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* [10]. 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 used to stop all or a portion of the testing activities [9]. 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 with respect to the likelihood and consequence scales. Our method consists of four main steps as illustrated in Fig. 1; dashed document icons represent input prepared during test planning, solid document icons represent output from one step and acts as input to the following step.

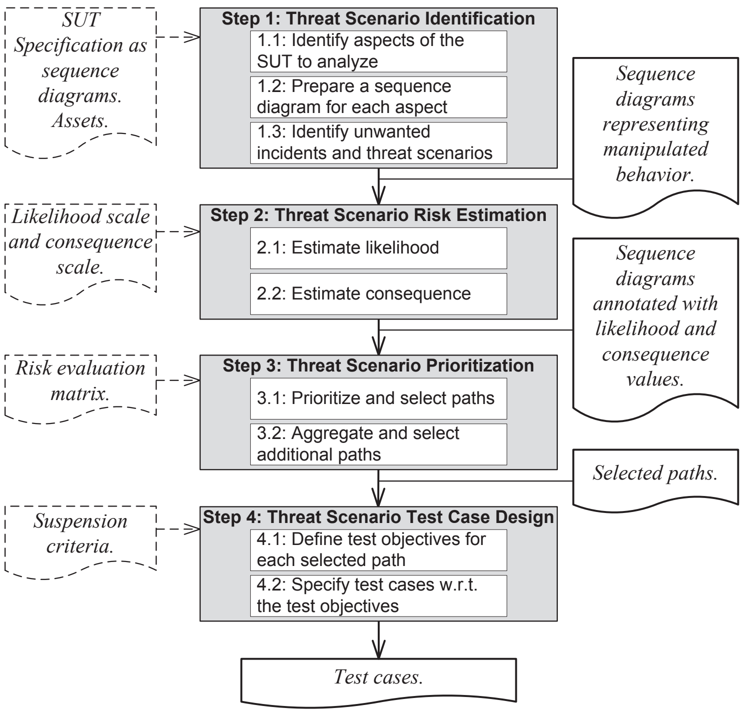

SUT Specification as sequence diagrams. Assets.
Step 1: Threat Scenario Identification
1.1: Identify aspects of the SUT to analyze
1.2: Prepare a sequence diagram for each aspect
1.3: Identify unwanted incidents and threat scenarios
Sequence diagrams representing manipulated behavior.
Likelihood scale and consequence scale.
Step 2: Threat Scenario Risk Estimation
2.1: Estimate likelihood
2.2: Estimate consequence
Sequence diagrams annotated with likelihood and consequence values.
Risk evaluation matrix.
Step 3: Threat Scenario Prioritization
3.1: Prioritize and select paths
3.2: Aggregate and select additional paths
Selected paths.
Suspension criteria.
Step 4: Threat Scenario Test Case Design
4.1: Define test objectives for each selected path
4.2: Specify test cases w.r.t. the test objectives
Test cases.

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

In **Step 1**, we analyze the SUT 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.

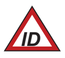
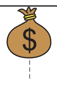
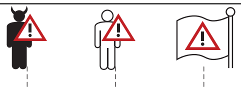
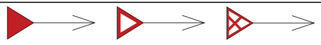
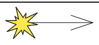
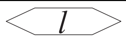
In **Step 2**, we estimate the likelihood for the occurrence of the threat scenarios and the unwanted incidents in terms of frequencies, the conditional probability for threat scenarios leading to other threat scenarios or to unwanted incidents, as well as the impact of unwanted incidents on the asset. 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. A sequence of threat scenarios may lead up to one or more risks. Additionally, different sequences of threat scenarios may lead up to the same risk. We refer to a sequence of threat scenarios leading up to and including a risk as a *path*.

In **Step 3**, we prioritize and select paths for testing. The input for this step is the output of Step 2. Additionally, this step employs the predefined risk evaluation matrix. **First**, we prioritize the paths by mapping them to the risk evaluation matrix based on the likelihood (frequency) value and the consequence (impact) value of the risk included in the paths. We then select the paths based on their risk level, i.e., their position in the risk evaluation matrix. **Second**, we aggregate similar risks in different paths in order to evaluate whether to select additional paths for testing. The output of this step is a set of paths selected for testing.

In **Step 4**, we define test objectives for each path selected for testing, and then we specify test cases with respect to the test objectives. A test objective is a textual specification of a well-defined target of testing, focusing on a single requirement or a set of related requirements as specified in the specification of the system under test [16]. A test objective merely describes what (logic) needs to be tested or how the system under test is expected to react to particular stimuli [16]. 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 define one or more test objectives for each path selected for testing. A path may have one or more test objectives, but one test objective is defined only for one path. We use one test objective as a basis for specifying one test case. **Second**, we specify a test case by first identifying the necessary interaction in the relevant path. By necessary interaction, we mean the interaction that is necessary in order to fulfill the test objective. Then, we copy the necessary interaction into a new sequence diagram. Finally, we annotate the new sequence diagram, with respect to the test objective, using the UML Testing Profile [16]. We continue designing test cases in this manner with respect to the predefined suspension criteria. 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 [12], we distinguish between three types of threats; deliberate threats (the leftmost lifeline in the Notation column), accidental 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 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.

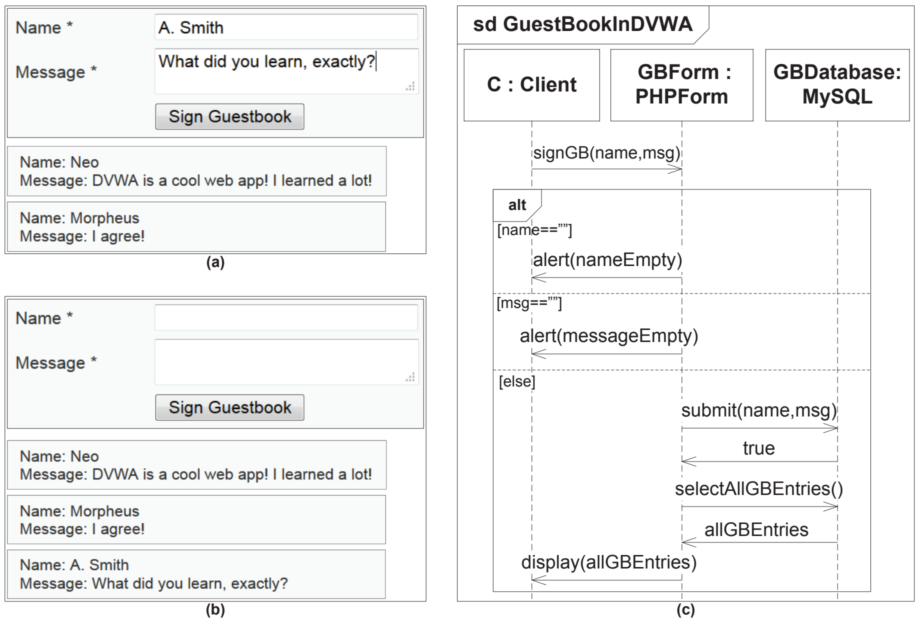
Table 1. Notation for annotating sequence diagrams with risk information.

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	$\boxed{p}$
Consequence	N/A	$\boxed{c}$

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) [4]. One of DVWA's main goals is to be an aid for security professionals to test their skills and tools in a legal environment [4]. 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 [26], which provides the required execution environment.

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.


Name * A. Smith
Message * What did you learn, exactly?
Sign Guestbook

Name: Neo
Message: DVWA is a cool web app! I learned a lot!

Name: Morpheus
Message: I agree!

(a)

Name *
Message *
Sign Guestbook

Name: Neo
Message: DVWA is a cool web app! I learned a lot!

Name: Morpheus
Message: I agree!

Name: A. Smith
Message: What did you learn, exactly?

(b)

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.

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 the following two security assets:

- *Integrity of the guest-book's source code.*
- *Availability of the guest book entries.*

In Sect. 4.1, we identify threat scenarios with respect to the integrity of the guest-book's source code, while in Sect. 4.2, we identify threat scenarios with respect to the availability of the guest book entries.

4.1 Identifying Threat Scenarios with Respect to the Integrity of the Guest-Book's Source Code

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 an aspect is determined by the tester.

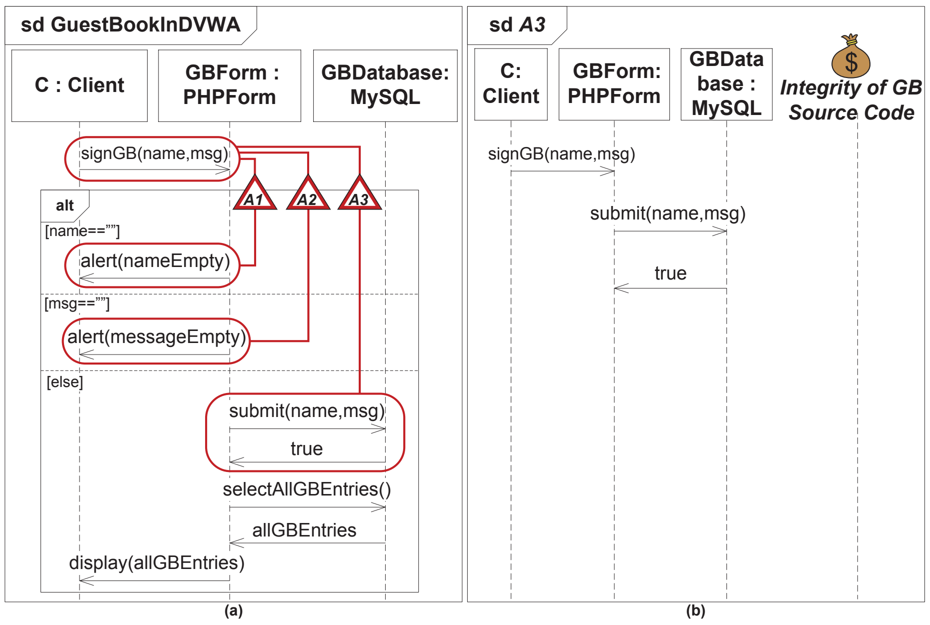


Fig. 3. **(a)** Specification of the guest book annotated with aspect labels. **(b)** Corresponding sequence diagram of the aspect encapsulated by aspect label A3, which also shows the security asset *integrity of the guest-book's source code* as a lifeline.

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. The integrity of the guest-book's source code may be 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 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) [18]. We identify the occurrence of an XSS script injection on the guest book database as an unwanted incident (*UI1*), as represented by the last message in Fig. 4. An XSS script is successfully injected in the guest book database only if the database successfully carries out the transaction containing the XSS script. This is why *UI1* occurs after the occurrence of message *true* on lifeline GBDatabase.

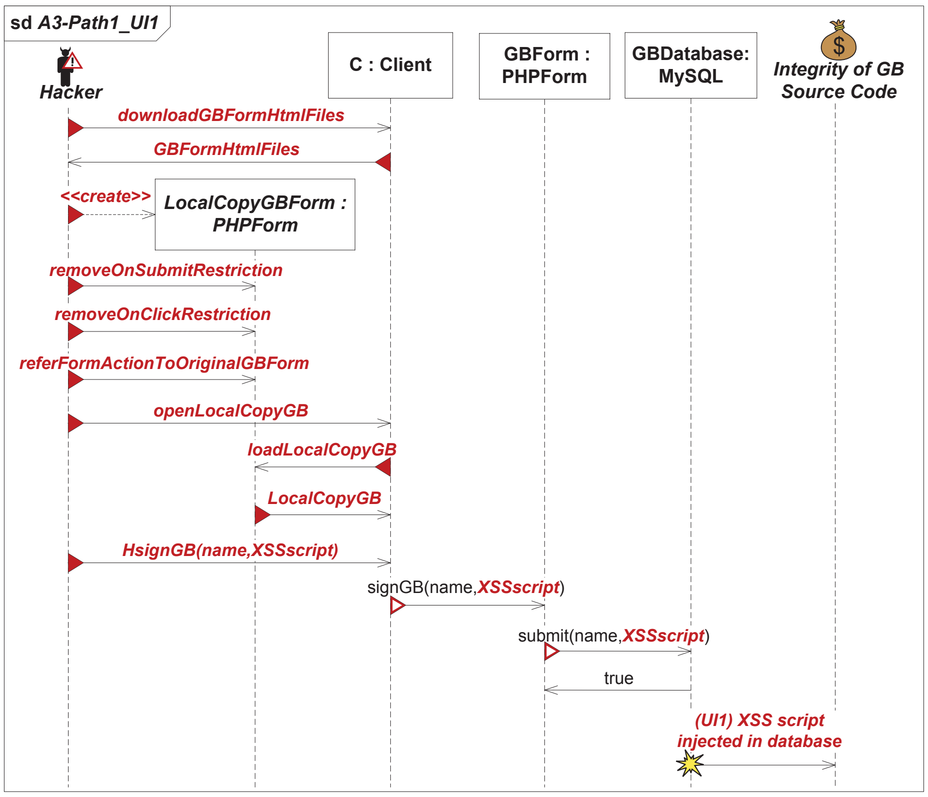


Fig. 4. Identifying a first path in which unwanted incident *UI1* may occur.

UI1 may be caused by different sequences of threat scenarios that manipulates the expected behavior of the guest book. Recall that we refer to a sequence of threat scenarios leading up to and including a risk as a path. In each of the sequence diagrams in Figs. 4, 5, and 6, we identify a different path in which *UI1* may occur.

The first path in which *UI1* may occur, i.e., the sequence diagram in Fig. 4, shows that *UI1* may occur if the *msg* parameter in messages *signGB(name, msg)* and *submit(name,msg)* is replaced with *XSSscript*, representing an XSS 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 script 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 script in the message field, and submits the guest book entry containing the XSS script (*openLocalCopyGB*, *loadLocalCopyGB*, *LocalCopyGB* and *HsignGB(name,XSSscript)*). Note that all of the messages described in this paragraph are annotated as new messages in the sequence diagram (message with a filled triangle).

The second path in which *UI1* may occur, i.e., the sequence diagram in Fig. 5, also shows that *UI1* may be caused by replacing the *msg* parameter in messages *signGB(name,msg)* and *submit(name,msg)* with *XSSscript*. However, we also see that the second path has some threat scenarios different from the first path, thus representing a different path in which *UI1* may occur.

In the second path, we first assume that the hacker gathers information about the setup of the URLs that are sent from a client to the guest book form. The hacker exploits this information to prepare a valid URL that contains an XSS script and that targets the guest book form. The process of preparing URLs in this way is also known as URL forging. This is commonly carried out by hackers, or other malicious users, with the objective to force legitimate users of a web application to execute actions on their behalf. These kinds of attacks are known as cross-site request forgery attacks [19]. Having successfully forged the URL containing the XSS script, the hacker sends it to a legitimate user of the guest book (*forgedURLReplacingMsgWithXSSscript*). We choose not to model how the hacker forges the URL and by what means the hacker sends the forged URL. The assumption is that a hacker successfully forges a URL capable of injecting an XSS script into the guest book database, and that the URL is successfully sent to a legitimate user of the guest book.

Having received the URL, the legitimate user executes it via the web browser of the client (*executeForgedURL*). Consequently, this results in the execution

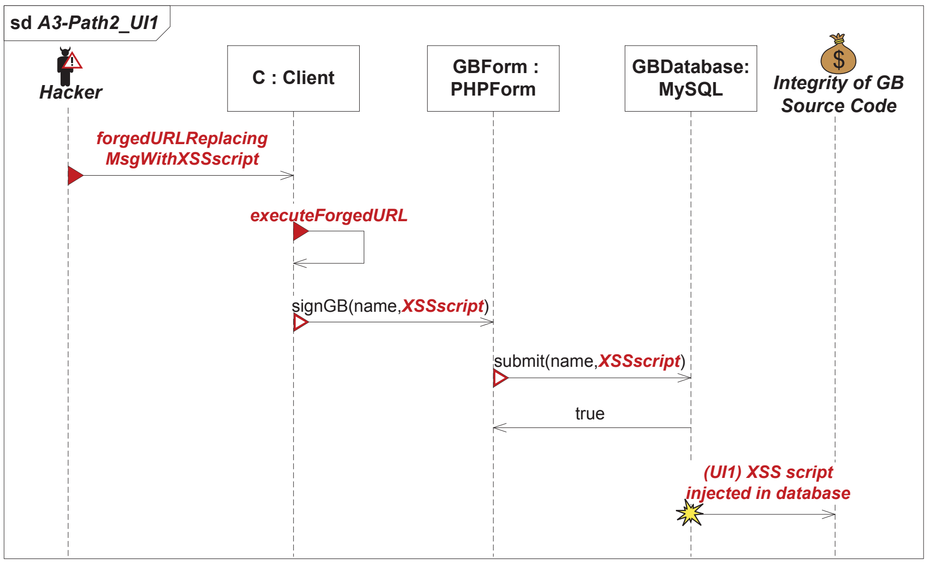


Fig. 5. Identifying a second path in which *UI1* may occur.

of the messages *signGB(name,XSSscript)*, *submit(name,XSSscript)* and *true*, which finally leads to the occurrence of *UI1*.

In the case of the third path, i.e., the sequence diagram in Fig. 6, we assume that the hacker is able to intercept the HTTPS connection between the client and the guest book form using a proxy tool by, for example, following the guidelines explained in [1]. The hacker first configures the tool to automatically inject an XSS script in a certain part of the HTTPS request sent to the guest book form (<<*create*>> and *configureAutoInjectXSSscriptInMsgInHTTPSRequest*). Then, the hacker starts the interception feature of the tool for intercepting the HTTPS request between the client and the guest book form (*interceptClientHTTPSRequest* and *interceptHTTPSRequest*). The consequence of intercepting the HTTPS requests sent from the client is the redirection of message *signGB(name,msg)* to the proxy tool. The redirection of message *signGB(name,msg)* is an alteration of the expected behavior of the guest book. Thus, we replace message *signGB(name,msg)* with a message representing an alteration.

Having successfully intercepted the HTTPS request sent from the client, the proxy tool automatically injects the XSS script into the HTTPS request (*injectXSSscriptInMsg*). Then, the proxy tool sends the HTTPS request containing the XSS script to the guest book form (*PTsignGB(name,XSSscript)*). Consequently, this results in the execution of the messages *submit(name,XSSscript)* and *true* as in the first and the second path, which finally leads to the occurrence of *UI1*.

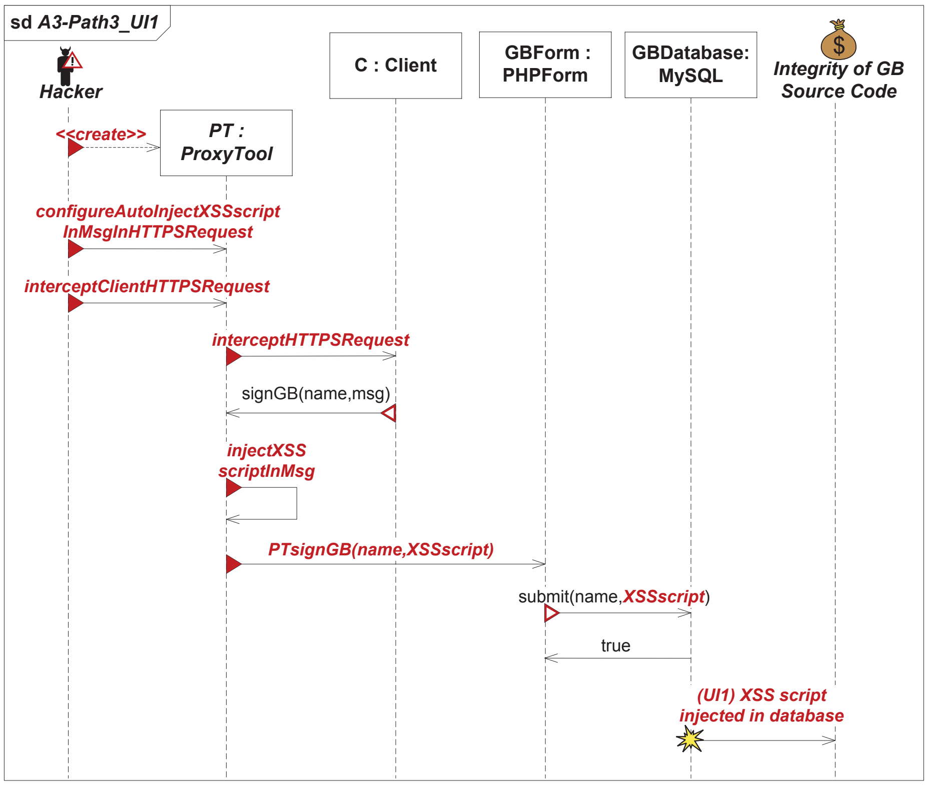


Fig. 6. Identifying a third path in which *UI1* may occur.

4.2 Identifying Threat Scenarios with Respect to the Availability of the Guest Book Entries

The identification of threat scenarios, with respect to the availability of the guest book entries, is carried out in a similar manner as explained in Sect. 4.1. As shown in Fig. 7a, we have identified one aspect labeled with aspect label B1. In this case, we are interested in analyzing the interaction composed of messages *signGB(name,msg)*, *submit(name,msg)*, *true*, *selectAllGBEntries()*, *allGBEntries* and *display(allGBEntries)*, with respect to the availability of the guest book entries. Figure 7b shows the sequence diagram corresponding to the interaction encapsulated by aspect label B1. We use the sequence diagram in Fig. 7b as a starting point for analyzing the SUT aspect encapsulated by aspect label B1, with respect to the availability of the guest book entries.

The availability of the guest book entries is compromised if, for example, the guest book entries in the guest book database are somehow deleted. The guest book entries may be deleted by executing an SQL query, on the guest book

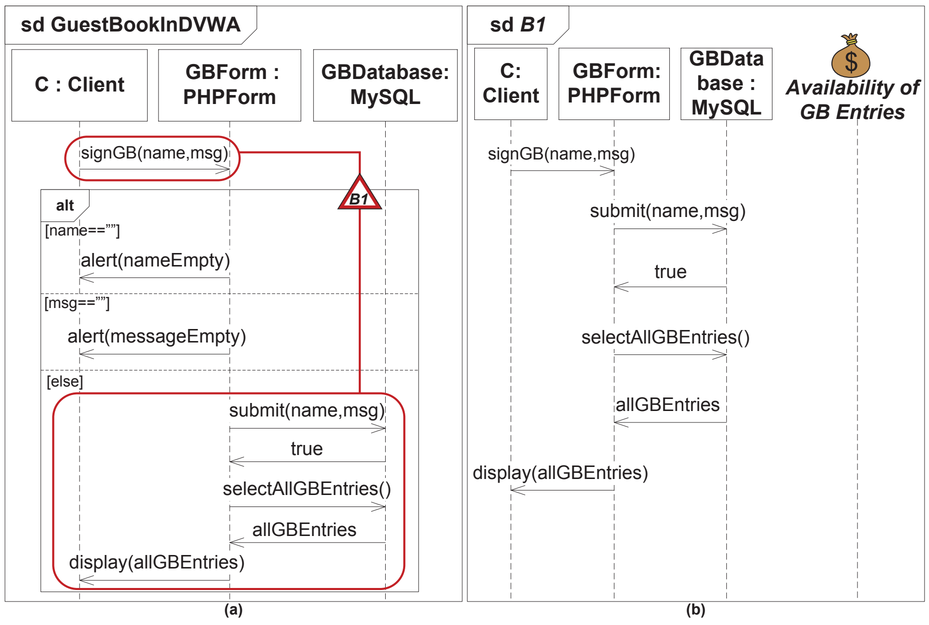


Fig. 7. (a) Specification of the guest book annotated with an aspect label. **(b)** Corresponding sequence diagram of the aspect encapsulated by aspect label B1, which also shows the security asset *availability of the guest book entries* as a lifeline.

database, which is constructed for deleting the guest book entries. Such SQL queries may be executed, for example, by submitting the queries to the database via the guest book form. This way of executing SQL queries is known as SQL injections. We identify this as unwanted incident *UI2*, as shown by message *(UI2) GB entries deleted due to SQL injection* in Fig. 8.

An SQL injection may be caused, from a hacker perspective, based on a similar path as the one presented in Fig. 4. The difference between the path in Fig. 4 and the path in Fig. 8 is that the hacker initiates an SQL injection (*HSignGB(name,SQLinjection)*), and that the *msg* parameter in messages *signGB(name, msg)* and *submit(name,msg)* in Fig. 8 is replaced with *SQLinjection*, representing an SQL query constructed for deleting guest book entries. Additionally, we see from Fig. 8 that the occurrence of unwanted incident *UI2* leads to some additional manipulations of the expected behavior of the guest book.

Given that unwanted incident *UI2* occurs, the guest book database no longer contains any guest book entries. However, having requested all guest book entries from the database (*selectAllGBEntries()*), the guest book form expects guest book entries. Naturally, the database does not return any guest book entries because there are none (*noGBEntries*), which in turn leads the guest book form not to display any guest book entries (*display(noGBEntries)*). These are al-

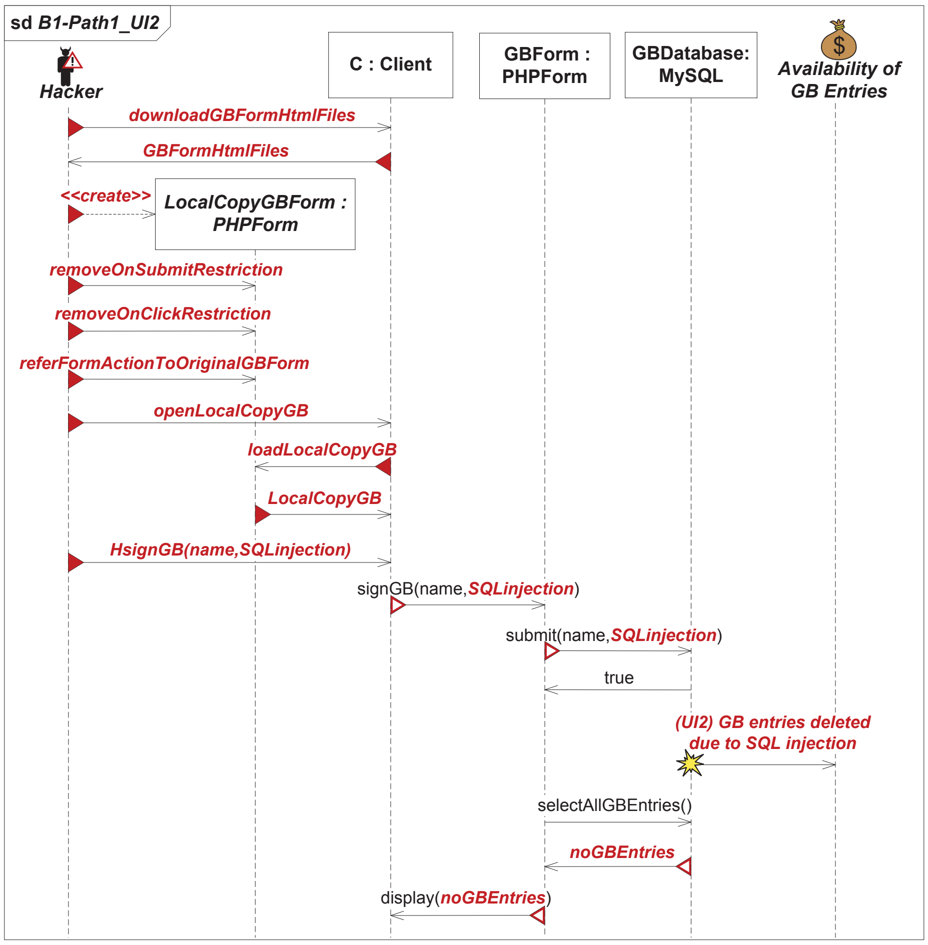


Fig. 8. Identifying a first path in which *UI2* may occur.

terations of the expected behavior of the guest book, as a result of the occurrence of *UI2*, and are therefore shown as messages representing alterations.

A second example of an unwanted incident, that compromises the availability of the guest book entries, is the deletion of the guest book entries before it reaches the client expecting them. This may be achieved by, for example, first intercepting the HTTPS response transmitted from the guest book form, and then deleting the guest book entries situated inside the captured HTTPS response. We identify this as unwanted incident *UI3*, as represented by message *(UI3) GB entries deleted by intercepting HTTPS response* in Fig. 9.

Similar to the third path in which *UI1* occurs, we assume that a hacker uses a proxy tool for intercepting the HTTPS connection between the guest book form and the client. The hacker first configures the tool for automatically deleting

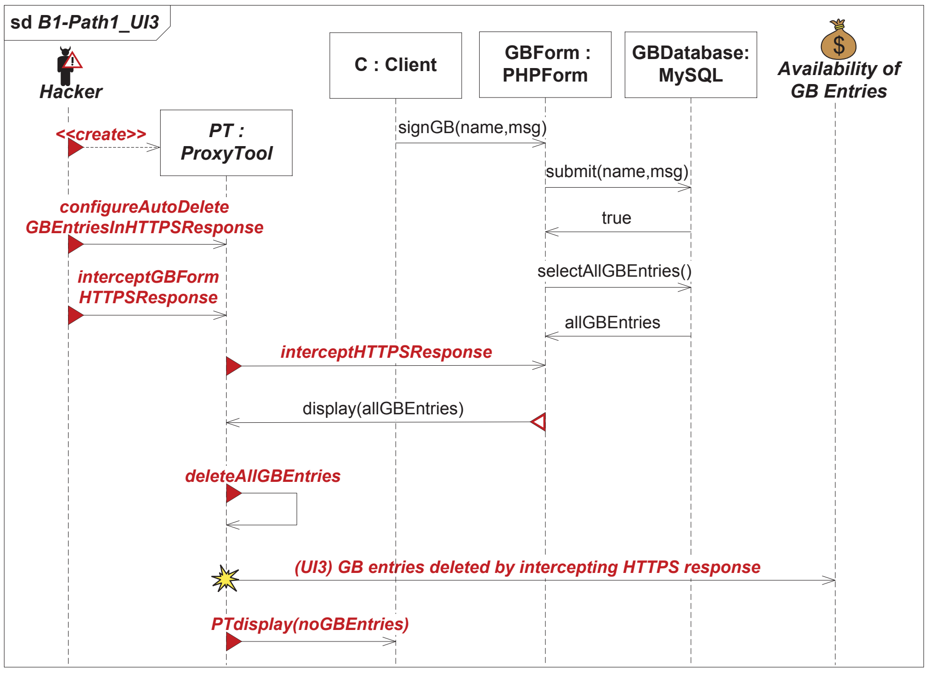


Fig. 9. Identifying a first path in which *UI3* may occur.

the guest book entries situated inside the HTTPS responses (<<*create*>> and *configureAutoDeleteGBEntriesInHTTPSResponse*). Then, the hacker starts the interception feature of the tool for intercepting the HTTPS response between the guest book form and the client (*interceptGBFormHTTPSResponse* and *interceptHTTPSResponse*). The consequence of intercepting the HTTPS responses sent from the guest book form is the redirection of message *display(allGBEntries)* to the proxy tool. The redirection of message *display(allGBEntries)* is an alteration of the expected behavior of the guest book. Thus, we replace message *display(allGBEntries)* with a message representing an alteration.

Having successfully intercepted the HTTPS response from the guest book form, the proxy tool automatically deletes all guest book entries situated in the HTTPS response (*deleteAllGBEntries*). This leads to the occurrence of unwanted incident *UI3*. Finally, the proxy tool sends the altered HTTPS response containing no guest book entries to the client (*PTdisplay(noGBEntries)*).

5 Step 2: Threat Scenario Risk Estimation

Table 2 shows the likelihood scale that we assume has been established during preparation of the risk analysis. The likelihood scale is given in terms of frequency intervals.

Table 2. Likelihood scale.

Likelihood	**Description**	
Rare	[0, 10>:1y	Zero to less than ten times per year
Unlikely	[10, 50>:1y	Ten to less than fifty times per year
Possible	[50, 150>:1y	Fifty to less than one hundred and fifty times per year
Likely	[150, 300>:1y	One hundred and fifty to less than three hundred times per year
Certain	[300, ∞>:1y	Three hundred times or more per year

In practice, it is common to use one likelihood scale when estimating the likelihood for the occurrence of threat scenarios and unwanted incidents. It is also possible to use one consequence scale, when estimating the consequence unwanted incidents have on certain assets. However, this may be difficult and impractical because the consequence unwanted incidents have on different assets may be difficult to measure by the same means. As mentioned in Sect. 4, we consider two different assets in this demonstration, namely the *integrity of the guest-book's source code* and the *availability of the guest book entries.*

Table 3 shows the consequence scale for security asset *integrity of the guest-book's source code.* The consequence scale in Table 3 is given in terms of impact on the integrity of certain categories of the guest-book's source code. 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 in Table 3. Table 4, on the other hand, shows the consequence scale for security asset *availability of the guest book entries.* The consequence scale in Table 4 is given in terms of impact on the availability of the guest book entries. For example, an unwanted incident has a catastrophic impact on the security asset if it makes the guest book entries unavailable for one week or more. Similar interpretations apply for the other consequences in Table 4. We assume that these consequence scales have been established during preparation of the risk analysis.

In Sects. 5.1 and 5.2, we make use of Table 2 for estimating the likelihood for the occurrence of threat scenarios and unwanted incidents. When estimating the consequence unwanted incidents have on the security assets, however, we make use of the consequence scale addressing the asset under consideration. That is, in Sect. 5.1 we use Table 3 and in Sect. 5.2 we use Table 4.

Table 3. Consequence scale for security asset *integrity of the guest-book's source code.*

Consequence	**Description**
Insignificant	The integrity of the source code that generates the aesthetics is compromised
Minor	The integrity of the source code that retrieves third party ads is compromised
Moderate	The integrity of the source code that generates the user interface is compromised
Major	The integrity of the source code that manages sessions and cookies is compromised
Catastrophic	The integrity of the source code that carries out database transactions is compromised

Table 4. Consequence scale for security asset *availability of the guest book entries.*

Consequence	**Description**
Insignificant	Guest book entries are unavailable in range [0, 1 minute>
Minor	Guest book entries are unavailable in range [1 minute, 1 hour>
Moderate	Guest book entries are unavailable in range [1 hour, 1 day>
Major	Guest book entries are unavailable in range [1 day, 1 week>
Catastrophic	Guest book entries are unavailable in range [1 week, ∞>

5.1 Estimating Risks Posed on the Integrity of the Guest-book's Source Code

Figure 10 shows likelihood estimates for the first path in which unwanted incident *UI1* occurs, as well as a consequence estimate for *UI1*. 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. 10. 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 script injection attacks are less likely to be initiated by hackers compared to other kinds of injection attacks they initiate (such as SQL-injection attacks) [22]. For this reason, we choose to assign a probability *0.8* on message *HsignGB(name,XSSscript)*, 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,XSSscript)*. 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

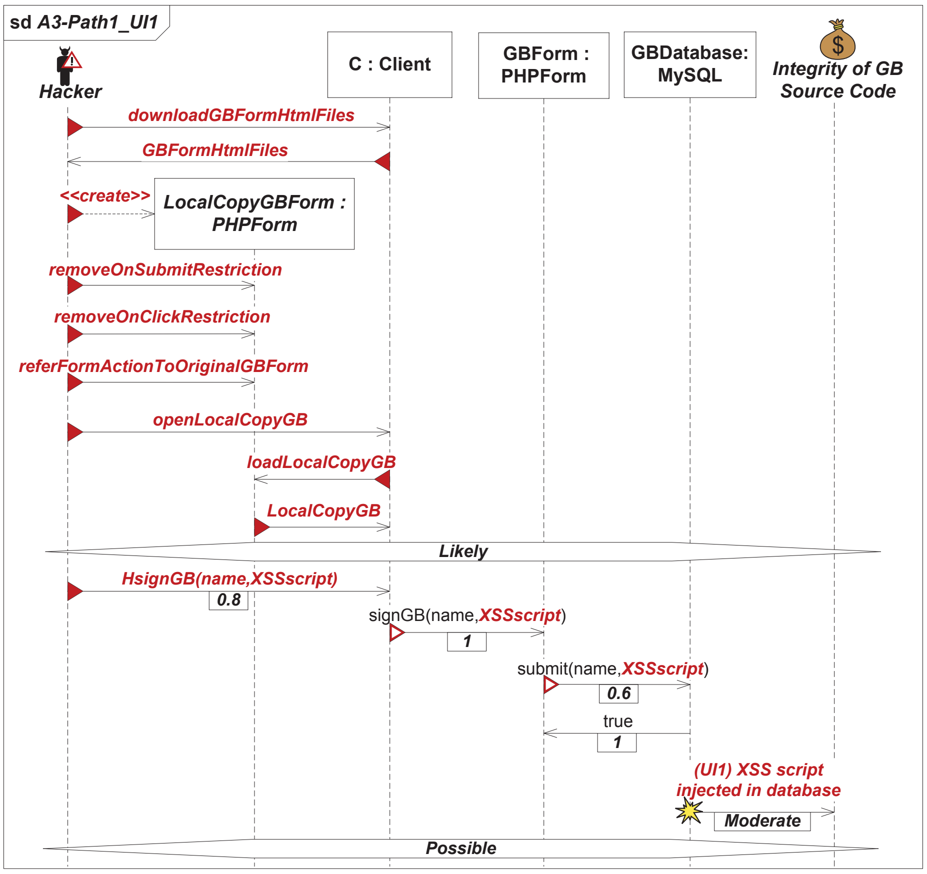


Fig. 10. Estimating the likelihood of the first path in which unwanted incident *UI1* occurs, as well as the consequence of *UI1*.

frequency interval, in the path, in a similar manner. This procedure is carried out until the frequency interval for the whole path 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 included in the path.

We proceed the estimation by identifying conditional probabilities for the remaining messages. We assume message *signGB(name,XSSscript)* 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 [23]. 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,XSSscript)*. We assume message *true* will occur with probability *1*, as there is nothing that prevents the database from executing the query containing the XSS script 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 the path represented by the sequence diagram in Fig. 10, and thereby unwanted incident *UI1*, may occur with likelihood Possible. Finally, an XSS script injected in the database has the objective to execute a script on the end user's web browser for different purposes. This means that the injected XSS script modifies the source code that generates the user interface. Thus, *UI1* has an impact on the security asset with a moderate consequence.

Figure 11 shows likelihood estimates for the second path in which *UI1* occurs. As explained in Sect. 4.1, the second path shows an example of how the hacker may inject an XSS script in the guest book database by performing a cross-site request forgery attack. The detection of whether a web application is vulnerable to cross-site request forgery attacks is easy [19], and thus an attack hackers most likely will try to exploit. Based on this, we assign likelihood Likely on the interaction composed of message *forgedURLReplacingMsgWithXSSscript*. However, the increased awareness of cross-site request forgery attacks has, in turn, brought about an increased usage of countermeasures preventing successful execution of such attacks [21]. For this reason, we choose to assign a probability *0.5* on message *executeForgedURL*.

Given that the forged URL is successfully executed, then there is nothing preventing the client in submitting the guest book entry containing the XSS script (*signGB(name,XSSscript)*). The probability for the occurrence of message *signGB(name,XSSscript)* is therefore *1*. The probability for the occurrence of messages *submit(name,XSSscript)* and *true* is *0.6* and *1*, respectively, for the same reason as given for the first path. The likelihood value for the second path is calculated in a similar way as explained for the first path. That is, we multiply the frequency interval *[150, 300>:1y* (likelihood Likely) with the product of the conditional probabilities assigned on the messages succeeding the likelihood assignment in the path, which in this case is *0.3*. This results in the frequency interval *[45, 90>:1y*. By mapping this frequency interval to the likelihood scale in Table 2, we see that it overlaps Unlikely and Possible. However, we also see that the frequency interval is skewed more towards Possible than Unlikely. For this reason, we choose to assign Possible on the second path, which means that *UI1* may occur with likelihood Possible in the second path. If a frequency interval overlaps several likelihood values, as it does for the second path, then the tester has to decide on which likelihood value to assign. In this demonstration, we decide to assign a likelihood value by analyzing the skewness of the frequency

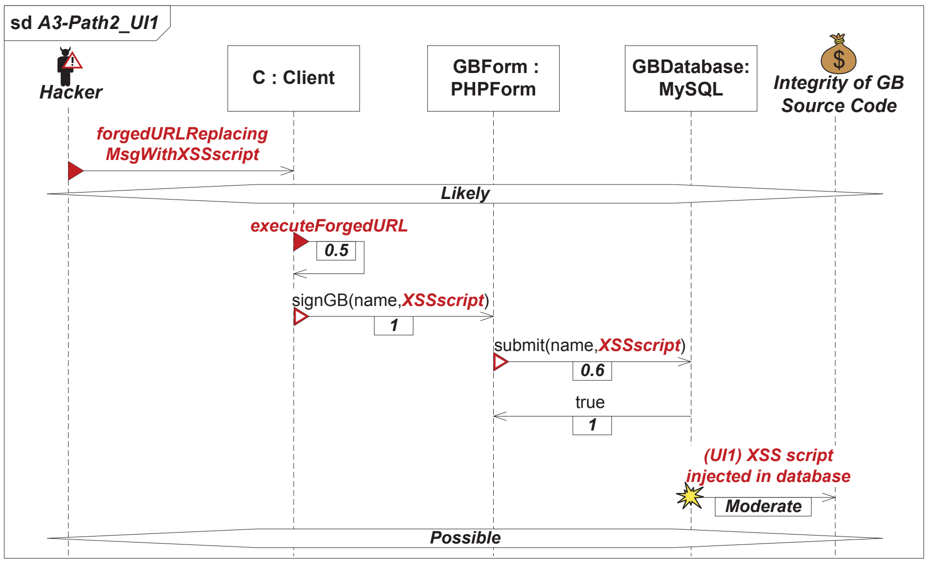


Fig. 11. Estimating the likelihood of the second path in which *UI1* occurs.

interval. Such decisions may vary from situation to situation and has to be made and justified by the tester.

Figure 12 shows likelihood estimates for the third path in which *UI1* occurs. Intercepting HTTPS connections is possible and in some situations easy to carry out [1]. However, the exploitability of vulnerabilities in encrypted communication protocols, such as HTTPS, is difficult on a large scale [20]. For this reason, we choose to assign likelihood Possible on the interaction starting with message *<<create>>* and ending with message *interceptHTTPSRequest*.

Let us, for the sake of the example, assume that the guest book is making use of proper countermeasures, e.g., as presented in [17], in order to significantly mitigate the possibility for successful HTTPS interceptions. Assuming this, we choose to assign a probability *0.2* on message *signGB(name,msg)*. If the HTTPS connection between the client and the guest book form is successfully intercepted, however, the proxy tool injects the *msg* parameter of message *signGB(name, msg)* with an XSS script (*injectXSSscriptInMsg*). Then, the proxy tool sends the guest book entry containing the XSS script to the guest book form (*PTsignGB(name,XSSscript)*). The probability for the occurrence of messages *injectXSSscriptInMsg* and *PTsignGB(name,XSSscript)* is therefore *1*. The probability for the occurrence of messages *submit(name,XSSscript)* and *true* is *0.6* and *1*, respectively, for the same reasons as given for the first path.

We calculate the likelihood value for the third path as explained for the first and the second path. That is, we multiply frequency interval *[50, 150>:1y* (likelihood Possible) with the product of the conditional probabilities assigned on the messages succeeding the likelihood assignment in the path, which in this

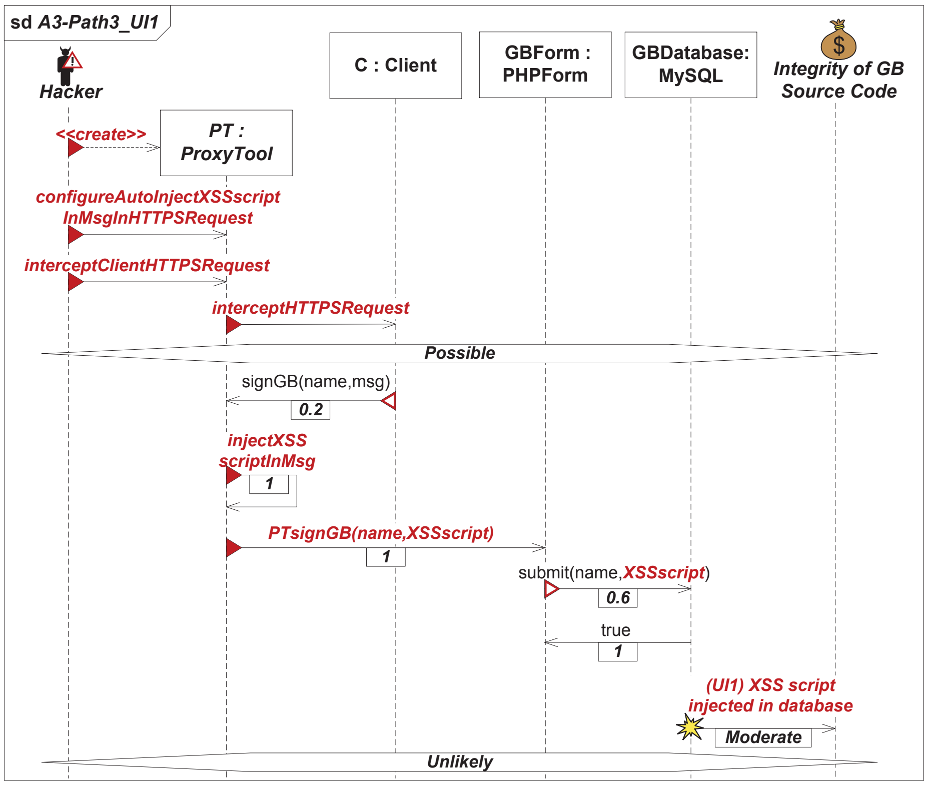


Fig. 12. Estimating the likelihood of the third path in which *UI1* occurs.

case is *0.12*. This results in the frequency interval *[6, 18>:1y*. We map this frequency interval to the likelihood scale in Table 2, and see that it is skewed more towards Unlikely than Rare. Based on this, we choose to assign likelihood Unlikely on the third path. Hence, *UI1* occurs with likelihood Unlikely in the third path.

5.2 Estimating Risks Posed on the Availability of the Guest Book Entries

As pointed out in Sect. 4.2, the path in which *UI2* occurs (see Fig. 13) is similar to the first path in which *UI1* occurs (see Fig. 10). In fact, the interaction starting with message *downloadGBFormHtmlFiles* and ending with message *LocalCopyGB* is identical in both paths. As shown in Fig. 10, we assigned likelihood Likely on the aforementioned interaction. Given that the aforementioned interaction is identical in both paths, we also assign likelihood Likely on the same interaction in the path where *UI2* occurs.

Hackers performing injection attacks will most likely carry out SQL-injection attacks [22]. We therefore choose to assign probability *1* on message *HsignGB(*

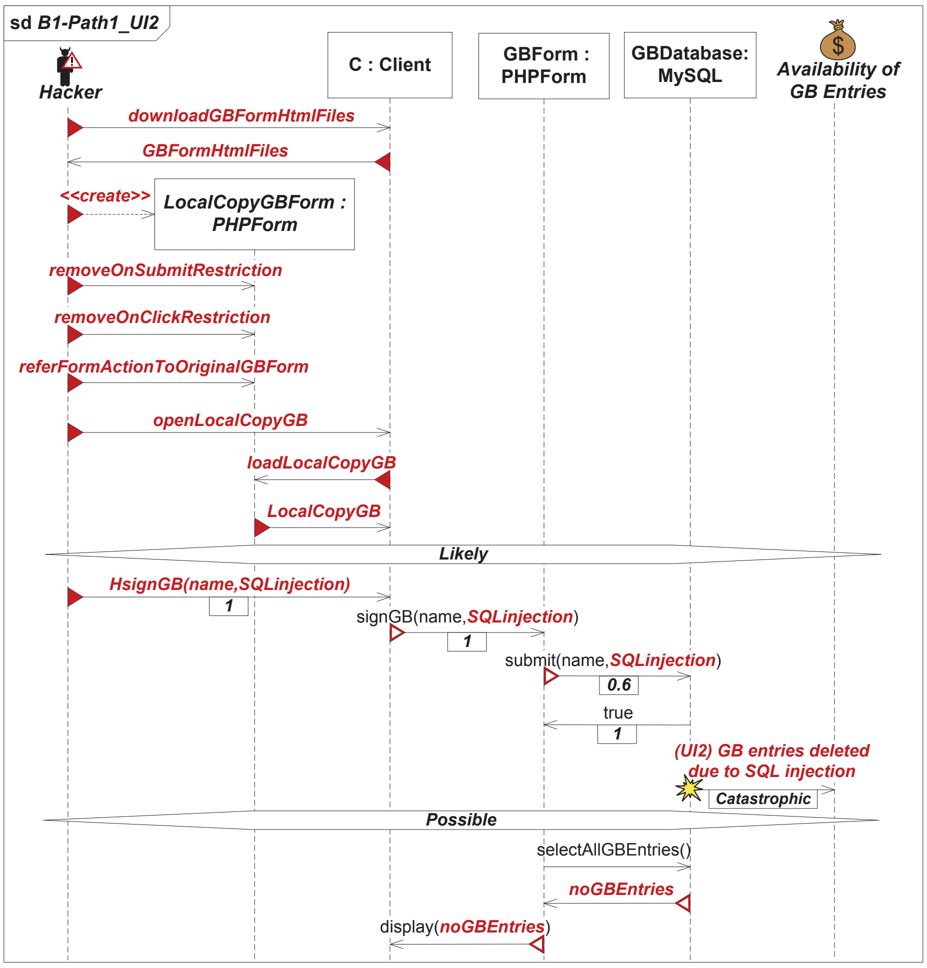


Fig. 13. Estimating the likelihood of the path in which *UI2* occurs, as well as the consequence of *UI2*.

name, SQLinjection). The probability for the occurrence of messages *signGB(name, SQLinjection)*, *submit(name,SQLinjection)* and *true* is *1*, *0.6* and *1*, respectively. The justification for assigning these three probability values is the same as the justification given for messages *signGB(name,XSSscript)*, *submit(name,XSSscript)* and *true* in the path shown in Fig. 10. Based on these conditional probabilities and likelihood Likely, we calculate the frequency interval for the whole path, i.e., the frequency interval for the occurrence of *UI2*, in a similar manner as explained throughout Sect. 5.1. The frequency interval for the occurrence of *UI2* is *[90, 180>:1y*, from which we have deduced likelihood Possible as shown in Fig. 13. Finally, the occurrence of *UI2* implies that the guest book entries in the database are deleted. Since the guest book in this demonstration does not have any mechanisms for creating a backup of the guest book entries, the deleted guest book entries will most likely never be available again. Thus, *UI2* has an impact on the security asset with a catastrophic consequence.

Figure 14 shows likelihood estimates for the path in which *UI3* occurs, as well as a consequence estimate for *UI3*. Similar to the third path in which *UI1* occurs (see Fig. 12), we assume that a hacker uses a proxy tool for intercepting the HTTPS connection between the client and the guest book form. Based on the same justification given for the third path where *UI1* occurs, we assign likelihood Possible on the interaction starting with message <<*create*>> and ending with message *interceptHTTPSResponse*.

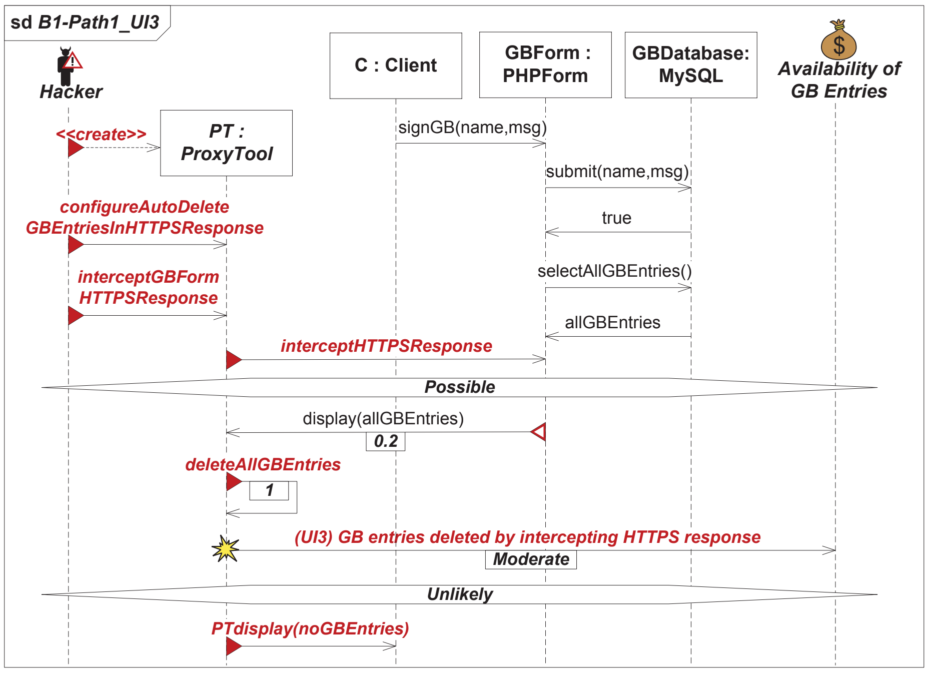


Fig. 14. Estimating the likelihood of the path leading to unwanted incident *UI3*, as well as the consequence of *UI3*.

Given that the guest book makes use of proper countermeasures against HTTPS interceptions, we assign probability *0.2* on message *display(allGBEntries)*. If, however, the HTTPS response gets intercepted, then there is nothing preventing the proxy tool from deleting the guest book entries situated inside the HTTPS response. Thus, we assign probability *1* on message *deleteAllGBEntries*. We calculate likelihood values as explained throughout Sect. 5.1 and see from Fig. 14 that *UI3* occurs with likelihood Unlikely. The occurrence of *UI3* implies that the guest book entries are deleted at an HTTPS response level. This means that the guest book entries are only deleted while in transit from the guest book to the client. Since the purpose of the guest book is to read and submit guest book entries, it is easily noticeable if the guest book constantly produces responses containing no guest book entries. Based on this observation, and because the guest book in this demonstration is rather simple and easy to administrate, one should be able to apply a fix within a day. Thus, *UI3* has an impact on the security asset with a moderate consequence.

6 Step 3: Threat Scenario Prioritization

Figure 15 shows the risk evaluation matrix established during preparation of the risk analysis. The risk evaluation matrix is composed of the likelihood scale in Table 2 and the consequence scale in Tables 3 and 4. 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 the setting of risk-driven testing, 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 Fig. 15 represents nine risk levels, horizontally 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 1, we identified five different paths. Three of these paths, i.e., the paths shown in Figs. 4, 5 and 6 includes *UI1*, which is a risk posed on the integrity of the guest-book's source code. Let us name these paths *UI1P1*, *UI1P2* and *UI1P3*, respectively. Similarly, let us name the path including *UI2* (see Fig. 8) as *UI2P1*, and the path including *UI3* (see Fig. 9) as *UI3P1*. *UI2* and *UI3* are risks posed on the availability of the guest book entries.

In Step 2, we estimated that *UI1P1* and *UI1P2* occur with likelihood Possible, and that *UI1P3* occurs with likelihood Unlikely. The risk caused by these paths, i.e., *UI1*, was estimated to have a moderate consequence on the integrity of the guest-book's source code. The likelihood for the occurrence of *UI2P1* was estimated to Possible, while the likelihood for the occurrence of *UI3P1* was estimated to Unlikely. Moreover, *UI2* and *UI3* were estimated to have a catastrophic and moderate consequence, respectively, on the availability of the guest book entries.

We map each path to the risk evaluation matrix with respect to the likelihood value and the consequence value of the risk included in the path. The result is shown in the risk evaluation matrix in Fig. 15. Let us say we are only interested in testing the paths that have a risk level `[5]` or higher. Based on this, we see from the risk evaluation matrix in Fig. 15 that we need to select *UI1P1*, *UI1P2* and *UI2P1* for testing. However, this selection excludes *UI1P3*, which is the third path that leads to *UI1*, and which is only one risk level less than the other

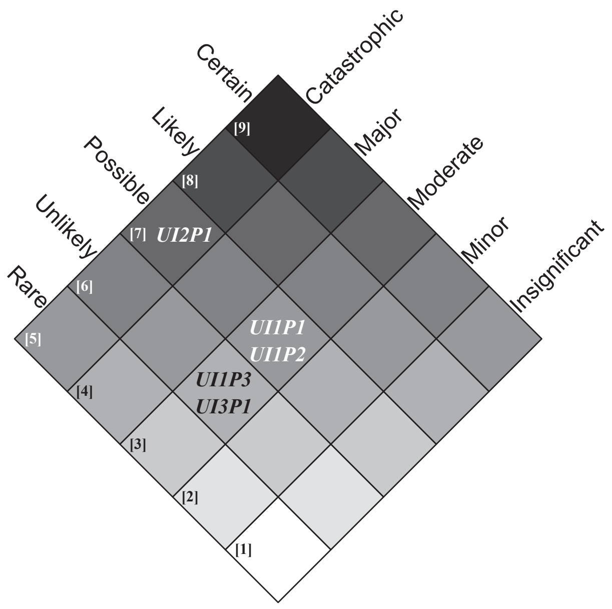


Fig. 15. Risk evaluation matrix.

two paths leading to *UI1*. Such clear cut selections are often difficult to justify because it is not obvious why some paths are selected for testing, while others are excluded. One way to come up with supporting evidence for confirming or refuting such clear cut selections, is to aggregate the likelihood values in the paths leading to the same risk.

UI1P1, *UI1P2* and *UI1P3* are separate paths. By separate paths, we mean paths that do not overlap in content such that no possible instance of one path can be an instance of the other. This also means that one path cannot be a special case of the other. We may therefore identify an aggregated likelihood value by summing up the frequency interval in each path. The frequency interval in *UI1P1*, *UI1P2* and *UI1P3* is *[72, 144>:1y*, *[45, 90>:1y* and *[6, 18>:1y*, respectively. We sum up these frequency intervals and get the new frequency interval *[123, 252>:1y*. We map this frequency interval to the likelihood scale in Table 2 and see that it is skewed more towards Likely than Possible. This means that the aggregated likelihood value for the paths *UI1P1*, *UI1P2* and *UI1P3* is Likely. However, we see from the frequency interval for *UI1P3* that it has an insignificant contribution for the aggregated likelihood value. In fact, we still get Likely as the aggregated likelihood value if we exclude the frequency interval for *UI1P3* from the aggregation. Because of this, we choose not to select *UI1P3* for testing. We select *UI1P1*, *UI1P2* and *UI2P1* for testing.

7 Step 4: Threat Scenario Test Case Design

Suppose, for the sake of the example, the following suspension criteria is given: "Define no more than two test objectives per path selected for testing, and specify a test case with respect to each test objective you define". The paths we selected for testing in Step 3 are *UI1P1*, *UI1P2* and *UI2P1*. The following lists one test objective for path *UI1P1*, one test objective for path *UI1P2*, and two test objectives for path *UI2P1*.

– **Test objective 1 for path *UI1P1***: Verify whether the guest book database (lifeline GBDatabase) stores an XSS script, by submitting an XSS script via the client (lifeline C).
– **Test objective 1 for path *UI1P2***: Verify whether the guest book database (lifeline GBDatabase) stores an XSS script by executing a forged URL, containing an XSS script, on the client (lifeline C).
– **Test objective 1 for path *UI2P1***: Verify whether the guest book form (lifeline GBForm) displays no guest book entries by submitting an SQL query, via the client (lifeline C), that is constructed for deleting the guest book entries.
– **Test objective 2 for path *UI2P1***: Verify whether the guest book database (lifeline GBDatabase) deletes guest book entries by submitting an SQL query, via the client (lifeline C), that is constructed for deleting the guest book entries.

We proceed by specifying test cases with respect to the test objectives. First, for each test objective, we identify the necessary interaction in the relevant path. By necessary interaction, we mean the interaction that is necessary in order to fulfill the test objective. Then, we copy the necessary interaction into a new sequence diagram. Finally, we annotate the new sequence diagrams, with respect to the test objectives, using the UML Testing Profile [16]. Because the tester defines the test objectives, it is the tester who knows which interactions are necessary to fulfill the test objectives. In test objective 1 for path *UI1P1*, we are interested in testing whether the guest book database stores an XSS script injected via the client. That is, we are interested in testing the interaction consisting of messages *signGB(name,XSSscript)*, *submit(name,XSSscript)* and *true* in path *UI1P1* (Fig. 4). Thus, we copy this interaction from path *UI1P1* into a new sequence diagram. The result is shown in Fig. 16a. Note that we choose not to copy other messages from path *UI1P1* because they are not needed for fulfilling the test objective.

In test objective 1 for path *UI1P2* (Fig. 5), we are interested in testing the interaction consisting of messages *executeForgedURL*, *signGB(name,XSSscript)*, *submit(name,XSSscript)* and *true*. In test objective 1 for path *UI2P1* (Fig. 8), we are interested in testing the interaction consisting of messages *signGB(name, SQLinjection)* and *display(noGBEntries)*. Finally, in test objective 2 for path *UI2P1* (Fig. 8), we are interested in testing the interaction consisting of messages *signGB(name, SQLinjection)*, *submit(name,SQLinjection)* and *true*. We

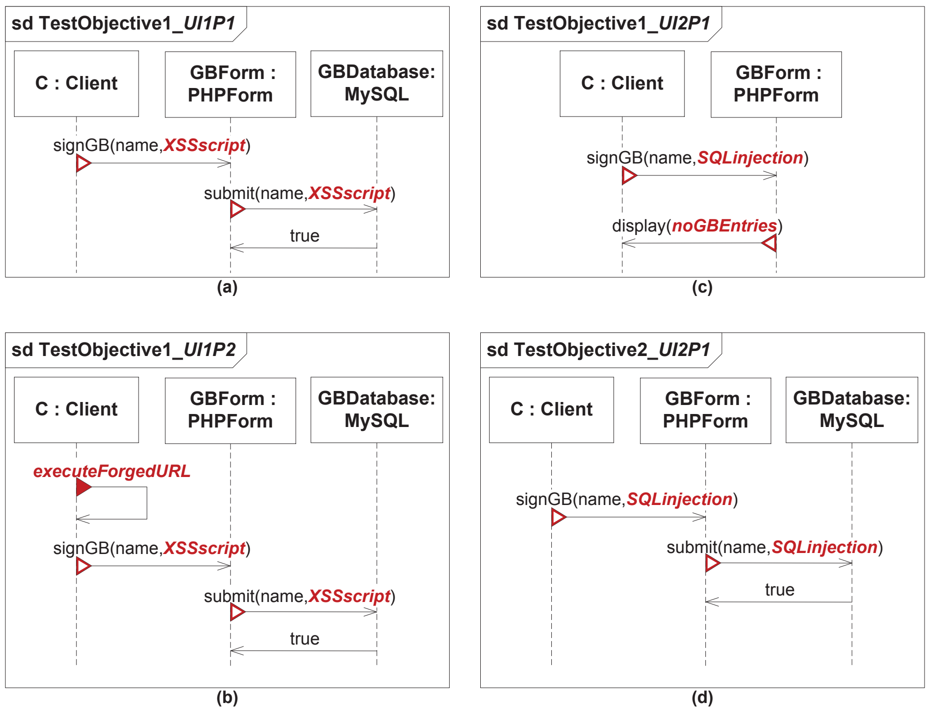


Fig. 16. **(a)** The interaction necessary to fulfill test objective 1 for path *UI1P1*. **(b)** The interaction necessary to fulfill test objective 1 for path *UI1P2*. **(c)** The interaction necessary to fulfill test objective 1 for path *UI2P1*. **(d)** The interaction necessary to fulfill test objective 2 for path *UI2P1*.

follow the same procedure as described above and model sequence diagrams containing the interactions necessary for fulfilling each of the test objectives in this paragraph. Figures 16b, 16c, and 16d show the interactions necessary for fulfilling test objective 1 for path *UI1P2*, test objective 1 for path *UI2P1*, and test objective 2 for path *UI2P1*, respectively.

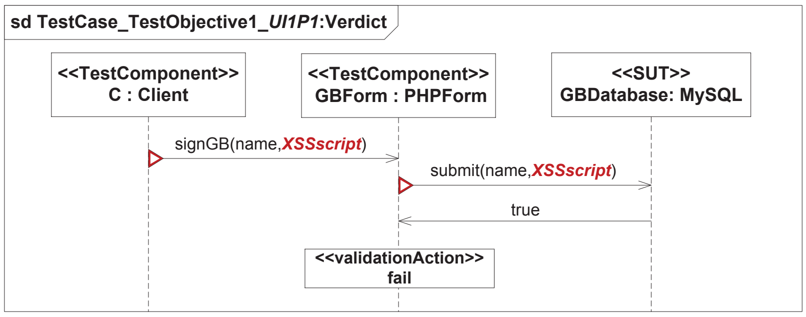
We specify test cases by annotating the sequence diagrams in Fig. 16 using the stereotypes given in the UML Testing Profile [16]: The stereotype <<SUT>> is applied to one or more properties of a classifier to specify that they constitute the system under test. The stereotype <<TestComponent>> is used 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.

The system under test in Fig. 16a is the guest book database (lifeline GB-Database), because we are testing whether the guest book database stores an XSS script submitted via the client. The system under test in Figs. 16b and 16d is also the guest book database. In the former, we again test whether the guest book database stores an XSS script, but this time we execute a forged URL containing an XSS script via the client. In the latter, we test whether the guest book database deletes the guest book entries by submitting an SQL query via the client. The system under test in Fig. 16c is the guest book form (lifeline GB-Form), because we are testing whether the guest book form displays no guest book entries as a result of executing an SQL injection. Based on this, we annotate lifeline GBDatabase in Figs. 16a, 16b and 16d, and lifeline GBForm in Fig. 16c with stereotype <<SUT>>.

The client (lifeline C) and the guest book form (lifeline GBForm) in Figs. 16a, 16b and 16d stimulate the system under test, i.e., the guest book database, with test data in terms of *XSSscript*, *XSSscript*, and *SQLinjection*, respectively. Thus, we annotate lifelines C and GBForm in Figs. 16a, 16b and 16d with stereotype <<TestComponent>>. In Fig. 16c, however, it is only the client that stimulates the system under test, which in this case is the guest book form. Thus, we annotate the client (lifeline C) in Fig. 16c with stereotype <<TestComponent>>.

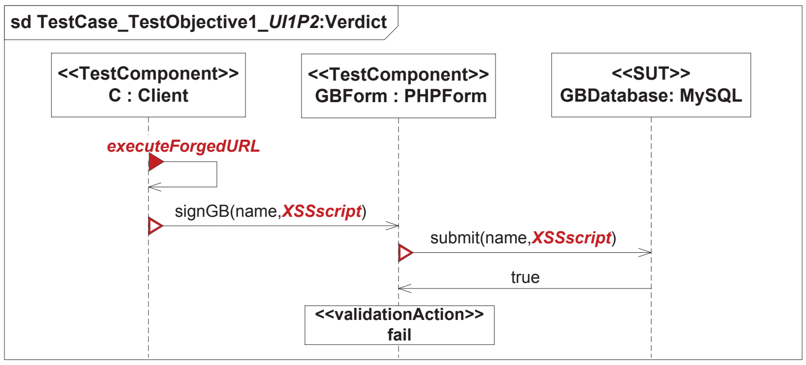
As mentioned above, test components are also used for evaluating whether the responses of the SUT adhere with the expected ones. We see from Figs. 16a, 16b and 16d that the test components receiving the responses of the SUT is the guest book form. Thus, we add an execution specification on lifeline GBForm in Figs. 16a, 16b and 16d, annotated with stereotype <<ValidationAction>> to set the verdict for the test case. Similarly, we add an execution specification on lifeline C in Fig. 16c, annotated with stereotype <<ValidationAction>>. The verdict is set to fail meaning that the SUT differs from the expected behavior. For example, if XSS script injection is successfully carried out then the SUT differs from the expected behavior, which should be to prevent XSS injections.

The outcome of these annotations is one test case, per test objective, as shown in Figs. 17, 18, 19 and 20. Figure 17 represents a test case specified with respect to test objective 1 for path *UI1P1*. Figure 18 represents a test case specified with respect to test objective 1 for path *UI1P2*. Figure 19 represents a test case specified with respect to test objective 1 for path *UI2P1*. Figure 20 represents a test case specified with respect to test objective 2 for path *UI2P1*.



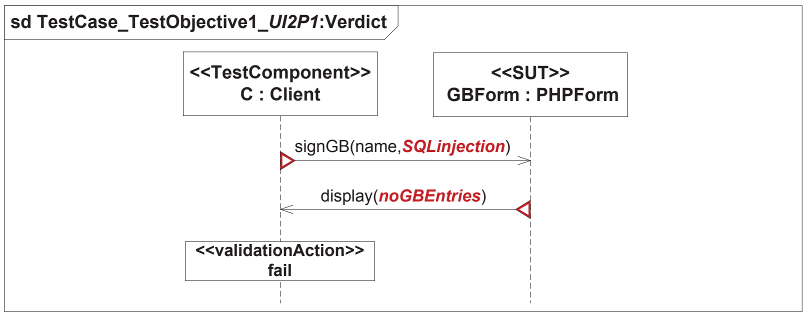
sd TestCase_TestObjective1_UI1P1:Verdict
<<TestComponent>> C : Client
<<TestComponent>> GBForm : PHPForm
<<SUT>> GBDatabase: MySQL
signGB(name,XSSscript)
submit(name,XSSscript)
true
<<validationAction>> fail

Fig. 17. Test case specified with respect to test objective 1 for path *UI1P1*.



sd TestCase_TestObjective1_UI1P2:Verdict
<<TestComponent>> C : Client
<<TestComponent>> GBForm : PHPForm
<<SUT>> GBDatabase: MySQL
executeForgedURL
signGB(name,XSSscript)
submit(name,XSSscript)
true
<<validationAction>> fail

Fig. 18. Test case specified with respect to test objective 1 for path *UI1P2*.



sd TestCase_TestObjective1_UI2P1:Verdict
<<TestComponent>> C : Client
<<SUT>> GBForm : PHPForm
signGB(name,SQLinjection)
display(noGBEntries)
<<validationAction>> fail

Fig. 19. Test case specified with respect to test objective 1 for path *UI2P1*.

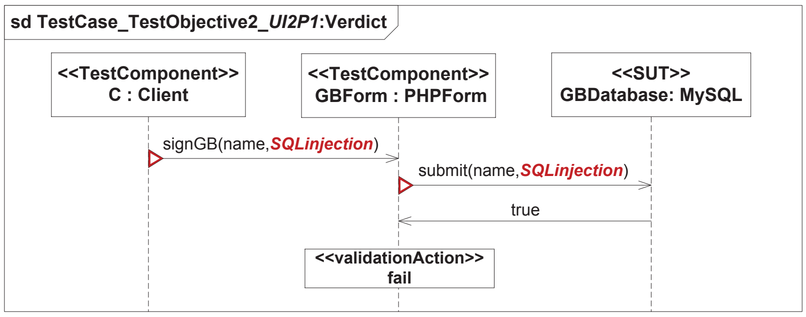

sd TestCase_TestObjective2_UI2P1:Verdict
<<TestComponent>> C : Client
<<TestComponent>> GBForm : PHPForm
<<SUT>> GBDatabase: MySQL
signGB(name,SQLinjection)
submit(name,SQLinjection)
true
<<validationAction>> fail

Fig. 20. Test case specified with respect to test objective 2 for path *UI2P1*.

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 [2, 11], risks are identified by making use of fault tree analysis, however, there is no explanation on how to estimate and prioritize the risks. In [7], the authors refer to fault tree analysis for identifying risks. There is no explanation on how to estimate and prioritize risks. In [13], the authors refer to a risk analysis approach published by NIST [25] 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 [27], 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 [3], 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 [24], risks are estimated by multiplying the probability that an entity contains fault with the associated damage. Similar to [3], 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 [2, 7, 11], 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 [13], 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 [27], 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 [3, 24], 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 [3], risks are estimated using tables, while in [24], 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 [10]. 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 [14]. Risk identification, estimation and prioritization in our method are in line with what is referred to as risk assessment in ISO 31000 [8]. Finally, our approach makes use of the UML Testing Profile [16] 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. F. Callegati, W. Cerroni, and M. Ramilli. Man-in-the-Middle Attack to the HTTPS Protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.
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. Y. Chen, R.L. Probert, and D.P. Sims. Specification-based Regression Test Selection with Risk Analysis. In *Proc. 2002 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'02)*, pages 1–14. IBM Press, 2002.

4. Damn Vulnerable Web Application (DVWA). http://www.dvwa.co.uk/. Accessed August 11, 2013.
5. G. Erdogan, Y. Li, R.K. Runde, F. Seehusen, and K. Stølen. Conceptual Framework for the DIAMONDS Project. Technical Report A22798, SINTEF Information and Communication Technology, 2012.
6. V. Garousi and J. Zhi. A survey of software testing practices in Canada. *Journal of Systems and Software*, 86(5):1354–1376, 2013.
7. 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.
8. International Organization for Standardization. *ISO 31000:2009(E), Risk management – Principles and guidelines*, 2009.
9. International Organization for Standardization. *ISO/IEC/IEEE 29119-1:2013(E), Software and system engineering - Software testing - Part 1: Concepts and definitions*, 2013.
10. International Organization for Standardization. *ISO/IEC/IEEE 29119-2:2013(E), Software and system engineering - Software testing - Part 2: Test process*, 2013.
11. 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.
12. M.S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis: The CORAS Approach.* Springer, 2011.
13. K.K. Murthy, K.R. Thakkar, and S. Laxminarayan. Leveraging Risk Based Testing in Enterprise Systems Security Validation. In *Proc. 1st International Conference on Emerging Network Intelligence (EMERGING'09)*, pages 111–116. IEEE Computer Society, 2009.
14. 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.
15. Object Management Group. *Unified Modeling Language (UML), superstructure, version 2.4.1*, 2011. OMG Document Number: formal/2011-08-06.
16. Object Management Group. *UML Testing Profile (UTP), version 1.2*, 2013. OMG Document Number: formal/2013-04-03.
17. R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12):2238–2246, 2006.
18. Open Web Application Security Project (OWASP). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS). Accessed September 5, 2013.
19. Open Web Application Security Project (OWASP). https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF). Accessed December 16, 2013.
20. OWASP Top 10 2013 – A6 – Sensitive Data Exposure. https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure. Accessed December 17, 2013.
21. OWASP Top 10 2013 – Release Notes. https://www.owasp.org/index.php/Top_10_2013-Release_Notes. Accessed September 6, 2013.
22. OWASP Top 10 Application Security Risks – 2013. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Accessed September 6, 2013.

23. PHP manual. http://php.net/manual/en/pdo.prepared-statements.php. Accessed September 6, 2013.
24. H. Stallbaum, A. Metzger, and K. Pohl. An Automated Technique for Risk-based Test Case Generation and Prioritization. In *Proc. 3rd International Workshop on Automation of Software Test (AST'08)*, pages 67–70. ACM, 2008.
25. G. Stoneburner, A. Goguen, and A. Feringa. Risk Management Guide for Information Technology Systems. NIST Special Publication 800-30, National Institute of Standards and Technology, 2002.
26. XAMPP. http://www.apachefriends.org/en/xampp.html. Accessed August 11, 2013.
27. 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.