SnT

securityandtrust.lu

Software Verification and Validation Laboratory: Modeling Security and Privacy Requirements for Mobile Applications: a Use Case-driven Approach

Xuan Phu Mai, Arda Goknil, Lwin Khin Shar, and Lionel C. Briand

Interdisciplinary Centre for Security, Reliability and Trust

University of Luxembourg

TR-SNT-2017-3

ISBN-13: 978-2-87971-160-7

July 07, 2017

Version 1.0

uni.lu

UNIVERSITÉ DU LUXEMBOURG

Modeling Security and Privacy Requirements for Mobile Applications: a Use Case-driven Approach

Xuan Phu Mai, Arda Goknil, Lwin Khin Shar, Lionel C. Briand
SnT Centre, University of Luxembourg, Luxembourg
{xuanphu.mai, arda.goknil, lwinkhin.shar, lionel.briand}@uni.lu

***Abstract*—Defining and addressing security and privacy requirements in mobile apps is a significant challenge due to the high level of transparency regarding users' (private) information. In this paper, we propose, apply, and assess a modeling method that supports the specification of security and privacy requirements of mobile apps in a structured and analyzable form. Our motivation is that, in many contexts including mobile app development, use cases are common practice for the elicitation and analysis of functional requirements and should also be adapted for describing security requirements. We integrate and adapt an existing approach for modeling security and privacy requirements in terms of security threats, their mitigations, and their relations to use cases in a misuse case diagram. We introduce new security-related templates, i.e., a mitigation template and a misuse case template for specifying mitigation schemes and misuse case specifications in a structured and analyzable manner. Natural language processing can then be used to automatically detect and report inconsistencies among artifacts and between the templates and specifications. Since our approach supports stakeholders in precisely specifying and checking security threats, threat scenarios and their mitigations, it is expected to help with decision making and compliance with standards for improving security. We successfully applied our approach to industrial mobile apps and report lessons learned and results from structured interviews with engineers.**

I. Introduction

Nowadays, mobile applications (or apps) are widely-used in many of our daily activities such as e-commerce and social networking. The widespread use of mobile apps brings a plethora of security and privacy risks, including malware that steals consumer and corporate data from mobile devices and mobile apps that unintentionally expose confidential data [1]. Therefore, security and privacy has become a crucial concern in mobile app development, starting from requirements analysis to implementation and testing.

The work presented in this paper is part of the EDLAH2 [2] project focusing on developing mobile apps in the healthcare domain. The project brings academic institutions and companies together in a consortium to enhance the lifestyle of elderly people through *gamification-based* apps. Gamification transforms activities that we normally hesitate to do, e.g., exercising regularly, into a competition [3]. EDLAH2's objectives entail that app users provide access not only to their personal data but also to their daily activities. Hence, software engineers must face the significant challenge of defining and ensuring security and privacy requirements in a context where there is a high level of transparency regarding users' (private) data.

In EDLAH2's business context, like in many others, use cases are the main artifacts employed to elicit functional requirements and communicate with stakeholders. Consequently, the usage scenarios describing how elderly people are expected to interact with the EDLAH2 apps is documented in use case diagrams and specifications. Therefore, to achieve widespread applicability, the need for integrating security requirements with use case modeling warrants the development of a use case-driven, security requirements modeling method that is, in our context, tailored to mobile app development.

Considerable research has been devoted to eliciting and analyzing security requirements using various forms of use cases (e.g., abuse cases [4] [5], security use cases [6], misuse cases [7] [8] [9] [10]). However, the applicability of these approaches in the context of security and privacy requirements modeling for mobile apps shows limitations with respect to (1) their support for explicitly specifying various types of security threats (a security threat is a possible event that exploits a vulnerability of the system to cause harm), (2) the definition of threat scenarios (a threat scenario is a flow of events containing interactions between a malicious actor and system to cause harm), and (3) the specification of mitigations for these threats.

These features are essential in the type of business context we target where it is required to explicitly identify the threat scenarios that may affect important business operations in order to identify appropriate mitigation schemes and trade-offs between functional requirements and security and privacy concerns. It is also expected that such security requirements, to be preferably specified in a structured and analyzable form, provide support for security testing, for example by helping with identification of attack surfaces (points at which security attacks can be executed). In addition to specifying security threats, a common practice in many environments requires mitigation schemes to be documented for the stakeholders to demonstrate compliance with applicable security and privacy standards and regulations. However, existing approaches lack reusable templates to specify such mitigation schemes.

In the context of modeling security and privacy requirements of mobile apps, the goal of this paper is to address the above challenges by proposing a use case-driven, security requirements modeling method called *Restricted Misuse Case Modeling (RMCM)*, which adapts existing methods and extends them. In RMCM, we employ misuse case diagrams proposed by Sindre and Opdahl [9] to model security and privacy

requirements in terms of use cases. Misuse cases model attacks that may compromise use cases and security use cases describe how to mitigate such attacks. For eliciting security threats and threat scenarios in a structured, and analyzable form, we adopt the Restricted Use Case Modeling method (RUCM) proposed in [11]. RUCM is based on a template and restriction rules, reducing ambiguities and incompleteness in use cases. It was previously evaluated through controlled experiments and has shown to be usable and beneficial with respect to making use cases less ambiguous and more amenable to precise analysis and design [12] [13] [14] [15] [16]. However, since RUCM was not originally designed for modeling security and privacy requirements, we extend the RUCM template with new restriction rules and constructs, targeting the precise modeling of security threats with a focus on mobile apps. Further, we provide a template for mitigation schemes and 3 mitigation schemes that are pre-specified with standard and secure coding methods for mitigating common security threats in mobile apps. They can be readily used and revised as necessary.

Leveraging on the analyzable form of our models, RMCM employs Natural Language Processing (NLP) to report inconsistencies between a misuse case diagram and its RMCM specifications, and to analyze the compliance of such specifications against the provided RMCM templates. NLP is also used to identify and highlight the control flow leading to different threat scenarios and the steps in RMCM specifications that refer to interactions between malicious actors and the system. The latter provides security testers with information about attack surfaces on which security testing should focus. To summarize, the contributions of this paper are:

- RMCM, a security requirements modeling method supporting the precise and analyzable specification of security threats, threat scenarios, and their mitigations, in the context of use case driven mobile app development;
- a practical toolchain, available at our tool website [17], including (1) a component that extends Papyrus [18] to support misuse case diagrams, (2) a component that extends IBM Doors [19] to support misuse case specifications and mitigation schemes in the RMCM templates, and (3) a component relying on NLP to detect inconsistencies among these artifacts;
- a case study demonstrating the applicability of RMCM in a realistic mobile app development context.

This paper is structured as follows. Section II introduces the context of our case study to provide the motivations behind RMCM. Section III discusses the related work. Section IV provides an overview of RMCM. Section V focuses on the use case extensions in RMCM. In Section VI, we present our tool support. Section VII reports on our industrial case study, from which we draw conclusions on the benefits and applicability of the proposed approach. Section VIII concludes the paper.

II. Context and Motivation

The context in which we developed RMCM was that of mobile apps, designed to engage users and enable interactions among them, accessing, processing, keeping track of user data and activities, and providing services. This entails the system we consider has a client-server database architecture, in which mobile apps interact with Web services for providing online activities, Web services interact with databases, and mobile apps interact with mobile device data storage such as SQLite and a SD card for storing data offline. In cases like EDLAH2, the system may consist of a Web portal interacting with Web services or even third-party software (e.g., other mobile apps).

Such mobile apps are representative of contexts where stakeholders face the significant challenge of defining and ensuring security and privacy requirements when there is a high level of transparency regarding users' information and the information flow across users and apps, including third-party software. There are various types of security threats that can harm such apps, such as information disclosure, information modification, unauthorized access, and denial of service. Each security threat can be further sub-classified according to the types of vulnerabilities being exploited. For example, information disclosure may arise from exploiting insecure data storage/flows, insecure authorization, SQL injection (SQLI), or cross site scripting (XSS) vulnerabilities. Hence, it is crucial to have such threats made explicit in the requirements specifications to systematically analyze the risks, help determine countermeasures to mitigate them, and demonstrate compliance with relevant security standards.

In this paper, we use the EDLAH2 system as a case study to motivate, illustrate, and assess RMCM. The EDLAH2 system provides a set of gamification-based functionalities on mobile devices that engage and challenge clients (older adults) to improve their physical, mental, and social activities. It allows the carers of the clients to create user accounts and configure the system through a website called MiCare. The gamification-based functionalities rely on a set of third-party mobile apps.

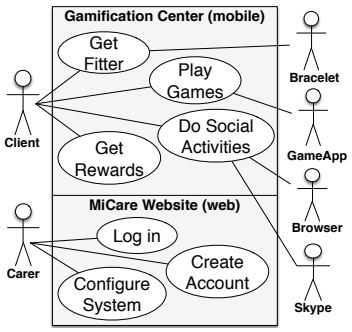

Fig. 1. Part of the use case diagram of EDLAH2

The current development practice in EDLAH2 is use case-driven and involves UML use case diagrams and use case specifications for describing functional requirements. Fig. 1 depicts part of the use case diagram of EDLAH2. *Client* and *Carer* are the main actors of the system while *Bracelet*, *Game App*, *Browser* and *Skype* are the secondary actors representing the third-party apps. The use cases describe seven main functionalities: *get fitter*, *play games*, *do social activities*, *get rewards*, *log in*, *create account*, and *configure system*.

A use case specification contains detailed description of a use case given in a use case diagram and usually conforms to a template [20] [21] [22]. The Cockburn template [20] had been followed so far to document EDLAH2 use case specifications in EDLAH2. Fig. 2 shows two examples of such specifications that are part of EDLAH2. *Log in* describes how the carer logs

into the system via the MiCare website. *Get Fitter* describes how the client checks his physical activities and condition (e.g., heart beat rate, number of steps and minutes of walking) as measured by the wearable device (i.e., bracelet).

```
1  USE CASE Log in
2  Precondition The system displays the login screen.
3  Basic Path
4  1. The system displays the login screen in a browser.
5  2. The carer enters the user name and password in the login form.
6  3. The system checks in the browser if the user name and password are valid.
7  4. The system builds a database query using the user name and password.
8  5. The system evaluates the query in the database.
9  6. The system checks that the query is successful.
10 7. The system displays the welcome message.
11 Postcondition The carer has successfully logged in the system.
12 Alternative Paths
13 3a. The entered user name or password is invalid.
14    3a1. The system displays the wrong user name or password message.
15 6a. The query is unsuccessful.
16    6a1. The system displays the database error message.
17
18 USE CASE Get Fitter
19 Precondition The client account has already been created.
20 Basic Path
21 1. The client requests to get current measurement.
22 2. The system receives the heart beat rate data from the bracelet.
23 3. The system checks whether the received heart beat rate data is correct.
24 4. The system stores the received data.
25 5. The system sets one point as a reward for the client.
26 6. The system displays the received heart beat rate data.
27 7. The system displays one point as a reward.
28 Postcondition The heart beat rate data and reward have been stored.
29 Alternative Paths
30 1a. The client requests to get activity data.
31    1a1. The system receives the client's activity data from the bracelet.
32    1a2. The system checks whether the activity data is correct.
33    1a3. The system stores the activity data.
34    1a4. The system sets two point as a reward for the client.
35    1a5. The system displays the activity data.
36    1a6. The system displays two points as a reward.
37     1a3a. The system displays the error for incorrect activity data.
38 4a. The system displays the error for incorrect heart beat rate measurement.
```

Fig. 2. Sample use case specifications for part of the EDLAH2 system

```
1  MISUSE CASE Get Unauthorized Access
2  Precondition Some client accounts have already been created in the system.
3  Basic Path
4  1. The crook tampers with the values in the login URL.
5  2. The crook submits the tampered URL directly to the system.
6  3. The system builds a query using the values provided in login URL.
7  4. The system evaluates the query in the database.
8  5. The system checks that the query is successful.
9  6. The system displays the welcome message.
10 Postcondition The crook has gained some privileges.
11 Alternative Paths
12 5a. The query is unsuccessful.
13    5a1. The system displays the database error message, revealing some
           information about the database structure.
14    5a2. The crook tampers with the values in the login URL again
           based on the exposed information.
15    5a3. The crook submits the tampered URL directly to the system
           until the system checks that the query is successful.
16 5a3. The crook reaches maximum number of login attempts.
17    5a3a. The system displays the error message for login.
18 Mitigation Points
19 mp1. In Step 3, the system sanitizes the values before building the query.
20 mp2. In Step 5a1, the system does not replay the exact database error message
        and instead, it displays only non-confidential information.
21
22 MISUSE CASE Expose Information from Mobile
23 Precondition The mobile device has a malware installed.
24 Basic Path
25 1. The malware requests access to user data stored in the system.
26 2. The system accepts the request.
27 3. The system sends user data to the malware.
28 Postcondition The malware has obtained user's private information.
29 Alternative Paths
30 2a. The system rejects the request.
```

Fig. 3. Sample misuse case specifications for part of the EDLAH2 system

The sample use case specifications reflect scenarios in which the client and carer use the system as expected. From these scenarios, one may also observe that the system accesses, processes, propagates, and stores the user's private information. As a result, security and privacy concerns need to be elicited such that stakeholders can take appropriate actions where needed. Standard use case templates, such as Cockburn's, are insufficient to document security and privacy concerns in use case specifications. One state-of-the-art approach for eliciting security concerns, together with functional requirements, provides a misuse case specification template [7] [9] which extends a use case template with additional fields such as *misuse* and *mitigation point*. We applied this template and attempted to elicit some of the security and privacy concerns in Fig. 2, as shown in Fig. 3. The *Get Unauthorized Access* and *Expose Information from Mobile* misuse case specifications we targeted are similar to the example given in [9]. The basic and alternative paths in Fig. 3 describe the sequence of actions that malicious actors and the system go through to cause harm. The mitigation points document the actions in a path where the misuse case can be mitigated (Lines 19-20 in Fig. 3).

Based on this attempt, we identified three challenges that need to be considered when capturing security and privacy requirements in use case-driven development of mobile apps:

Eliciting security threats in an explicit, precise form (*Challenge 1*). We identified that although the template we applied supports specifying various threats, it does not support their specification in a precise and unambiguous manner. This is the same for other related approaches such as [9] [7] [6] [4]. First, the templates do not provide glossary or keywords for specifying common security threats in mobile apps. For instance, the *Get Unauthorized Access* misuse case in Fig. 3 corresponds to unauthorized access via SQLI (for categorizing the security threats for mobile apps, we follow the common, well-known terminology given in OWASP [23]). In the specification, the term "SQL" was not even used. Likewise, the *Expose Information from Mobile* misuse case corresponds to information disclosure due to insecure data storage. Second, the templates do not explicitly distinguish between malicious actor-system interactions and other types of interactions. For instance, the steps in Lines 4-5 in Fig. 3 correspond to the malicious actor-system interactions, whereas the steps in Lines 6-9 correspond to the system's internal state changes. The interactions between malicious actors and the system contain information about the attack surfaces. But since the specification provided in Fig. 3 does not make this important difference, it may not be straightforward for a security tester to precisely determine where the attack surface is. In this case, the attack surface is the parameters in the login URL.

The templates also do not provide a precise and systematic way to determine malicious actors in the specification. However, providing security extensions or keywords for precisely specifying common security threats in mobile apps would be useful. It would facilitate unambiguous communication among stakeholders and support various automated analyses, including security testing (e.g., identify where a security attack may come from and from whom).

Eliciting threat scenarios in a structured form (*Challenge 2*). The existing templates have two shortcomings in eliciting threat scenarios. First, they do not have any explicit control flow structure (e.g., *Do-While*, *Do-Until*, and *IF-Then*). For instance, the *Get Unauthorized Access* misuse case in Fig. 3 tries a list of user name and password tuples iteratively until the malicious user logs into the system to get privileges. Since we do not have any explicit loop structure in the template we use for misuse cases in Fig. 3, we tried to describe the loop condition for the threat in a non-restrictive natural language form ('...until the system checks that...' in Line 15 in Fig. 3). However, it is not clear where the iteration starts in the execution flow. Second, we need extensions for distinguishing different types of scenarios — (basic and alternative) scenarios that a malicious actor may follow to successfully harm the system and scenarios that may not result in such harm. For instance, in the *Get Unauthorized Access* misuse case in Fig. 3, there are two alternative paths — the one starting from Line 12 leads to the scenario where the malicious actor harms the system and the other one starting from Line 16 leads to the scenario where the malicious actor fails to harm the system. As a result, it may not be easy for the stakeholders or an analysis tool to distinguish control flows and conditions leading to threat scenarios. Therefore, such specifications can be ambiguous and cannot support automated analyses.

Eliciting mitigation schemes (*Challenge 3*). After identifying security threats in threat scenarios, it is crucial to specify mitigation schemes matching these threats to demonstrate that the software design complies with applicable security and privacy standards and regulations. Such mitigation schemes provide the developers with guidance on how to prevent security threats specified in misuse cases. Different security threats and threat scenarios often share common mitigation methods and guidance. For instance, the two different security threats — information disclosure via SQLI and unauthorized access via SQLI — can both be mitigated by parameterizing the SQL queries. Existing work only supports specifying the flow of events mitigating each specific threat scenario. Such flows of events are embedded in misuse case specifications where one should specify the mitigation points (Lines 18-20 in Fig. 3). There is no structured way to specify the guidance for developers to mitigate security threats. In other words, there is a lack of template support for specifying mitigation schemes that can be reused and adapted for various security threats.

In this work, we focus on how to best address these three challenges in a practical manner. Automated test generation for security testing is one potential application of our method, but we leave it out for future work.

III. Related Work

There are numerous approaches in the literature to model security and privacy requirements [24] [25] [26] [27] [28]. In a comprehensive literature review [24], security requirements engineering methods are distinguished across six categories: *multilateral* [29] [30], *UML-based* [9] [31], *goal-oriented* [32] [33], *problem frame-based* [34] [35], *risk/threat analysis-based* [36] [37], and *common criteria-based approaches* [38] [39]. Since our modeling method is based on misuse cases [9] extending UML use case diagrams, it can be considered a *UML-based approach*. It also relies on a form of *risk/threat analysis* to capture various security threats, to elicit threat scenarios in a structured form, and to specify mitigation schemes. There are previous works devoted to documenting security and privacy requirements in various forms of use cases (e.g., abuse cases [4] [5], security use cases [6], and misuse cases [7] [8] [9] [10]). We discuss those previous works with respect to the challenges identified in Section II.

Eliciting security threats in a precise form. McDermott and Fox [4] [5] propose *abuse cases* to describe harmful interactions (i.e., security threats) between a system and malicious actors, but relations between abuse cases and other types of requirements are not described. Sindre and Opdahl [9] extend the UML use case diagram with *misuse cases* and *security use cases* to model security threats (i.e., *misuse*) with security-related requirements (i.e., *threat mitigation*) and with other functional requirements. Alexander discusses automation to support misuse case diagrams [40] [41] and reports experiences with misuse case diagrams in an industrial setting [42]. Rosado et al. [43] show how misuse case diagrams are employed to model the security requirements of a grid application, while Rostad [8] extends misuse case diagrams with the notion of *vulnerability*, i.e., a weakness that may be exploited by misusers. Misuse case diagrams can be employed to represent misuse cases, security use cases and their relations, but not to capture security threats in misuse cases. Sindre and Opdahl [10] adapt a use case specification template for detailed textual descriptions of threat scenarios. This template is later extended for misuse case generalization [44] and reuse [45]. However, the template does not provide any construct or restriction rule to capture security threats in a precise and analyzable form to support automated analysis. Firesmith [6] proposes a similar template, suffering from the same shortcomings, for *security use cases* which represent security-related requirements combined with a form of threat scenarios. El-Attar [46] [47] proposes a structure, i.e., *Structured Misuse Case Descriptions (SMCD)*, that guides the analysts towards developing consistent misuse case diagrams and specifications. The proposed structure embeds some keywords, e.g., *misuse case* and *include*, within fields present in common use case templates. However, the keywords do not support capturing security threats in an explicit form. To the best of our knowledge, our work is the first that provides restriction rules and keywords (see Table II in Section V-B) to precisely capture security threats in misuse case specifications.

Eliciting threat scenarios in a structured form. The templates proposed for misuse cases [10], [46], [47], security use cases [6] and abuse cases [5] extend the common use case templates in the literature to elicit security requirements. But in general they do not provide any extension or any control flow structure to systematically identify and capture various threat scenarios. Whittle et al. [48] propose the use of sequence diagrams for the analysis of threat scenarios in

misuse cases. Though these diagrams come with control flow structures, their approach does not provide a systematic way to distinguish attack types and threat scenarios causing harm from those that don't. We extend the RUCM template because it already provides control flow structures, e.g., *'do...while'* and *'if...then...else...'*, which can also be used for security threat scenarios. We propose new extensions to capture success and failure scenarios in a structured and analyzable form.

Eliciting mitigation schemes. The template proposed by Sindre and Opdahl [10] supports mitigation points where one can specify the flow of events mitigating each specific threat scenario. There is, however, no structured way to specify mitigation schemes, i.e., the guidance and methods for developers to mitigate security threats. RMCM is the first that provides template support for specifying mitigation schemes, which can be reused for mitigating various security threats.

IV. Overview of Our Modeling Method

The process in Fig. 4 presents an overview of our modeling method. RMCM is designed to address the challenges stated above in the use case-driven development context we described for mobile apps, and builds upon and integrates existing work. The RMCM output is a misuse case diagram, use case specifications, security use case specifications, misuse case specifications, and mitigation schemes.

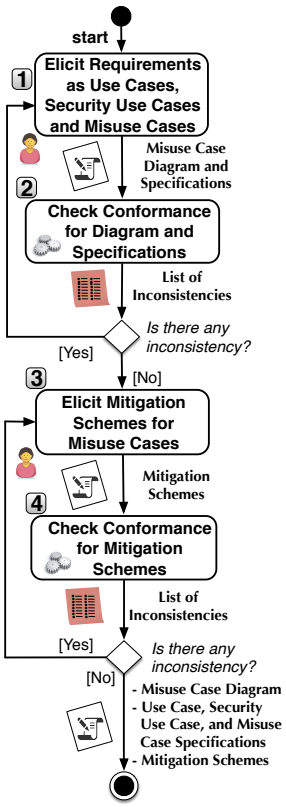

Fig. 4. Approach overview

In Step 1, *Elicit requirements as use cases, security use cases and misuse cases*, the analyst elicits functional and security requirements relying on a misuse case diagram and the extended RUCM (RMCM) template. Functional requirements and security requirements are captured in the misuse case diagram while it is further detailed in use case, security use case and misuse case specifications (*Challenges 1 and 2*). While use cases capture functional requirements, security use cases capture security countermeasures addressing potential attacks, which are themselves represented with misuse cases.

In Step 2, *Check conformance for diagram and specifications*, RMCM-V (Restricted Misuse Case Modeling - Verifier), the tool we developed for RMCM, automatically checks the consistency between the misuse case diagram and specifications, and between the specifications and the RMCM template. It relies on NLP. If there is any inconsistency, the analyst updates the diagram or specifications (Step 1). Steps 1 and 2 are iterative: the specifications and diagram are updated until the specifications conform to the RMCM template and they are consistent with the diagram. In Step 3, *Elicit mitigation schemes for misuse cases*, mitigation schemes are elicited for the security threats specified in misuse cases (*Challenge 3*).

In Step 4, *Check conformance for mitigation schemes*, RMCM-V automatically checks whether the mitigation schemes conform to the mitigation template. Steps 3 and 4 are also iterative: the mitigation schemes are updated until they conform to the template.

V. Capturing Security Requirements

In this section, we explain the artifacts produced by RMCM. We discuss how they were extended, compared to what was proposed in existing work, and illustrate how they address our three challenges with our running example.

A. Use Case Diagram with Misuse Case Extensions

To capture misuse cases, security use cases, use cases, and their relationships, RMCM relies on the misuse case extensions proposed by Sindre and Opdahl [9] for use case diagrams. We made this choice for RMCM because of the explicit representation of misuse cases, security use cases, and their relationships (i.e., *threaten* and *mitigate*). In the following, we briefly introduce our extensions. The reader is referred to [9] for further details. Fig. 5 depicts part of the misuse case diagram for EDLAH2.

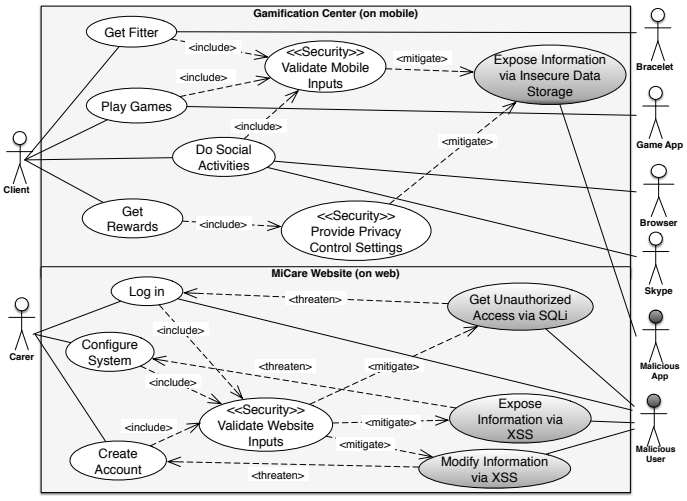

Fig. 5. Part of the misuse use case diagram for EDLAH2

As shown in Fig. 5, misuse cases, i.e., sequence of actions that a malicious actor can perform to cause harm, are greyed to distinguish them from use cases. Likewise, malicious actors (e.g., Malicious app) are distinguished from benign actors (e.g., Carer) and labeled with the keyword 'malicious'. The 'security' stereotype is used to distinguish security use cases that are countermeasures against misuse cases. In addition to the use case relationships (e.g., *include* and *extend*), *mitigate* is used for specifying the relationships between security use cases and misuse cases; and *threaten* is used for specifying the relationships between misuse cases and use cases [9]. For instance, in Fig. 5, *Validate Website Inputs* mitigates *Get Unauthorized Access via SQLI*, which threatens *Log in*.

Expose Information via Insecure Data Storage is an abstract misuse case that is extended by some concrete misuse cases threatening *Get Fitter*, *Play Games*, *Do Social Activities*, and *Get Rewards*. For space reasons, we do not show them here. Please refer to our website [17] for details.

B. Misuse Case and Security Use Case Specifications

Regarding misuse case specifications, to elicit security threats in a precise form and to elicit threat scenarios in a structured and analyzable form (*Challenges 1 and 2*), we propose the RMCM template, an extension of the RUCM template, shown in Table I, and new restriction rules, shown in Table II. The misuse case specifications are elicited using this template, further using the new restriction rules in addition to the original ones. These template and restriction rules are designed to make (mis)use case specifications explicit, precise, and analyzable by restricting the use of natural language and by using specific keywords. Our extensions specifically target the modeling of security and privacy concerns for mobile apps.

Fig. 6 shows two simplified misuse case specifications of EDLAH2 written in RMCM. The original RUCM provides basic and alternative flows which we adapted as *Basic Threat Flow*, *Specific/Bounded/Global Alternative Flow* and *Specific/Bounded/Global Alternative Threat Flow* (see Table I).

TABLE I
RESTRICTED MISUSE CASE MODELING (RMCM) TEMPLATE

Misuse Case Name	The name of the misuse case.	
Brief Description	Summarizes the misuse case in a short paragraph.	
Precondition	What should be true before the misuse case is executed.	
Primary Actor	The actor which initiates the misuse case.	
Secondary Actors	Actors which interact with the system to accomplish the misuse case.	
Dependency	Include and extend relationships to other (mis)use cases.	
Generalization	Generalization relationships to other misuse cases.	
Threats	Threaten relationships to use cases.	
Basic Threat Flow	Specifies the main sequence of actions that the misuser carries out to harm the system.	
	Steps(numbered)	Flow of events
	Postcondition	What should be true after the basic flow executes.
Specific/Bounded/Global Alternative Threat Flow	A specific alternative sequence of actions that the misuser carries out to harm the system.	
	RFS	A reference flow step number where flow branches from.
	Steps(numbered)	Flow of events
	Postcondition	What should be true after the alternative flow executes.
Specific/Bounded/Global Alternative Flow	A specific alternative sequence of actions that do not result in any harm to the system.	
	RFS	A reference flow step number where flow branches from.
	Steps(numbered)	Flow of events
	Postcondition	What should be true after the alternative flow executes.
Mitigation Scheme	Refers to the name of the mitigation scheme, specified using our mitigation template, to mitigate this misuse case. This complements security use case(s).	

TABLE II
RMCM EXTENSIONS

#	Description	Explanation
R1	MALICIOUS	Referring to a malicious actor to enforce explicitly describing the actions/steps that involve a malicious actor.
R2	DATA	Referring to the security-sensitive or privacy data to enforce explicitly describing the actions/steps that access or modify security-sensitive or privacy data.
R3	SQLI	Referring to SQL injection attacks to enforce explicitly describing the type of security threat.
R4	XPATHI	Referring to XPath injection attacks to enforce explicitly describing the type of security threat.
R5	XMLI	Referring to XML injection attacks to enforce explicitly describing the type of security threat.
R6	LDAPI	Referring to LDAP injection attacks to enforce explicitly describing the type of security threat.
R7	XSS	Referring to cross site scripting attacks to enforce explicitly describing the type of security threat.
R8	JSONI	Referring to JSON injection attacks to enforce explicitly describing the type of security threat.
R9	BO	Referring to Buffer overflow attacks to enforce explicitly describing the type of security threat.
R10	RME	Referring to remote code execution attacks to enforce explicitly describing the type of security threat.
R11	GETS ⟨data⟩ FROM ⟨location⟩	Enforcing explicitly describing the security threat that exploits insecure data storage or unintentional data flows vulnerabilities (e.g., the MALICIOUS app GETS credit card DATA FROM log files).
R12	PROVIDES ⟨attack⟩ VALUES IN ⟨parameter⟩	Enforcing explicitly describing the security threat that exploits injection vulnerabilities. ⟨attack⟩ is the parameter in which the injection attack type (listed in R3-R10) is to be specified explicitly (e.g., the MALICIOUS user PROVIDES SQLI VALUES IN name and password).
R13	BYPASSES ⟨service-request⟩ REQUEST TO ⟨server-program⟩	Enforcing explicitly describing the security threat that exploits insecure authentication or access control vulnerabilities, which allows malicious actors to bypass any direct interaction with the client program and directly submit service requests to the server program (e.g., the MALICIOUS app BYPASSES view users REQUEST TO viewInfo program).
R14	EXPLOITS ⟨error-message⟩	Enforcing explicitly describing the security threat exploiting information exposed in error or exception messages, which allows a malicious actor to understand the system better and conduct informed security attacks (e.g., the MALICIOUS user EXPLOITS exception message from the system).
R15	SENDS PRIVILEGED ⟨permission⟩ REQUEST TO ⟨client-program⟩	Enforcing explicitly describing the security threat exploiting insecure authorization schemes, which allows a malicious app to request the mobile app to execute privileged functionalities (e.g., the MALICIOUS app SENDS PRIVILEGED phone call REQUEST TO the main activity program).

Different from a basic flow in a use case specification, which describes a nominal scenario for an actor to use the system as intended, a basic threat flow describes a nominal scenario for a malicious actor to harm the system. It contains misuse case steps and a postcondition (Lines 5-12 and 36-39). A misuse case step can be one of the following interactions: a malicious actor initiates a security attack to the system (Lines 6, 7, 17, 18, 19 and 37); the system validates a request and/or data (Line 10); the system replies to a malicious actor with a result (Lines 11, 16, 26 and 38). A step can also capture the system altering its internal state (Lines 8 and 9). In addition, the inclusion of another use case can be specified as a step. All keywords are written in capital letters for readability.

Based on examples, the following discusses how the rules in Table II (*R1-R15*) are applied to address *Challenge 1*:

The step in Line 6 applies *R12* to explicitly specify the security threat in which a malicious actor, labelled with the 'MALICIOUS' keyword (*R1*), initiates an SQL injection attack through the two user input fields 'user name' and 'password' of the login URL. The 'SQLI' keyword (*R3*) is used to explicitly specify the type of security threat. The step

1 **MISUSE CASE** Get Unauthorized Access via SQLi
2 **Precondition** Some client accounts have already been created in the system.
3 **Primary Actor** MALICIOUS user
4 **Threats** Log In
5 **Basic Threat Flow**
6 1. The MALICIOUS user PROVIDES SQLI VALUES IN user name and password of the login url.
7 2. The MALICIOUS user BYPASSES the login REQUEST TO the login server program.
8 3. The system builds a query with the values provided in login url.
9 4. The system evaluates the query in the database.
10 5. The system VALIDATES THAT the query is successful.
11 6. The system SENDS the welcome message TO the MALICIOUS user.
12 **Postcondition** The MALICIOUS user gained an unauthorized access.
13 **Specific Alternative Threat Flow**
14 RFS 5
15 1. DO
16 2. The system SENDS the database error message DATA TO the MALICIOUS user.
17 3. The MALICIOUS user EXPLOITS the database error message DATA from the system.
18 4. The MALICIOUS user PROVIDES SQLI VALUES IN user name and password of the login url.
19 5. The MALICIOUS user BYPASSES the login REQUEST TO the login server program.
20 6. UNTIL the query is successful.
21 7. RESUME STEP 6.
22 **Postcondition** The MALICIOUS user gained an unauthorized access.
23 **Bounded Alternative Flow**
24 RFS SATF 1-6
25 1. IF the maximum number of login attempts is reached THEN
26 2. The system SENDS the invalid login message TO the MALICIOUS user.
27 3. ABORT.
28 4. ENDIF.
29 **Postcondition** The MALICIOUS user did not gain an unauthorized access.
30 **Mitigation Scheme** Secure Coding for Server-side Program
31
32 **MISUSE CASE** Expose Information via Insecure Data Storage
33 **Precondition** The mobile device, in which the system is installed, also has a MALICIOUS app installed. The client has already used the system.
34 **Primary Actor** MALICIOUS app
35 **Threats** Get Fitter, Play Games, Do Social Activities
36 **Basic Threat Flow**
37 1. The MALICIOUS app GETS user location DATA FROM the log file of the system.
38 2. The system SENDS the user location DATA TO the MALICIOUS app.
39 **Postcondition** The MALICIOUS app has obtained user's private information.
40 **Specific Alternative Flow**
41 RFS 2
42 1. IF user location DATA is encrypted THEN
43 2. ABORT.
44 3. ENDIF.
45 **Postcondition** The MALICIOUS app didn't obtain user's location information.
46 **Mitigation Scheme** Secure Coding for Mobile Program

Fig. 6. Misuse case specifications in RMCM

in Line 7 specifies another threat in which the malicious actor bypasses the input validation method, possibly implemented on the client side (browser), and submits the login URL to the login server program directly (*R13*). Notice that in place of the 'SQLI' keyword as the value of the parameter ⟨attack⟩ in *R12*, the keywords 'XPATHI', 'XMLI', 'LDAPI', 'XSS', 'JSONI', 'BO', 'RME' described in *R4-R10* can be used to explicitly elicit other types of code injection security threats for mobile apps. The step in Line 17 applies *R14* to specify a security threat that exploits the information exposed in error or exception messages, labelled with the keyword 'DATA' (*R2*). The step in Line 37 applies *R11* to specify a security threat in which a malicious actor attempts to access the user location data, labelled with the keyword 'DATA' (*R2*), locally stored in the mobile device (specified by stating the location of the data: 'log file of the system' in the ⟨location⟩ parameter).

In the following, we discuss with examples how *Challenge 2* is addressed. The 'VALIDATES THAT' keyword (Line 10), described in the original RUCM [11], indicates a condition that must be true to take the next step, otherwise an alternative flow is taken. It is one of the control flow structures we use for threat scenarios. In Fig. 6, the system proceeds to Step 6 (Line 11) if the query is successful (Line 10).

In the original RUCM template, there are three types of alternative flows: specific, bounded and global. In RMCM, we employ these alternative flows to describe failure scenarios for security attacks. An alternative flow always refers to and depends on a condition in a specific step of the basic threat flow. A bounded alternative flow refers to more than one step in the basic flow (Lines 23-29) while a global alternative flow refers to any step in the basic flow. For specific and bounded alternative flows, the keyword RFS is used to refer to one or more reference flow steps (Line 24).

In the RMCM template (Table I), we introduce Specific/Bounded/Global alternative *threat* flows to describe alternative success scenarios and to distinguish them from failure scenarios for security attacks. For instance, in the *Get Unauthorized Access via SQLI* in Fig. 6, the specific alternative threat flow describes another success threat scenario (Lines 13-22) where the query is not validated by the system in the basic threat scenario (Line 10). The bounded alternative flow (Lines 23-29) describes the failure scenario for the attack given in this alternative threat flow (Lines 13-22).

Bounded and global alternative (threat) flows begin with the 'IF .. THEN' keyword, which is described in the original RUCM template, to describe the conditions under which alternative (threat) flows are taken (Line 24). Specific alternative flows do not necessarily begin with 'IF .. THEN' since a guard condition can be indicated in its reference flow step (Line 10). In addition, to describe threat scenarios, we also use other control flow structures — 'DO...UNTIL' and 'MEANWHILE' — which are described in the original RUCM template. For instance, in the *Get Unauthorized Access via SQLI* misuse case in Fig. 6, the malicious user tries a list of user name and password tuples iteratively in an attempt to log in the system to obtain privileges. By having such explicit loop structure (Lines 15 and 20), we are able to specify where the iteration starts and ends in the execution flow of the threat scenario.

In RMCM, use case specifications are elicited according to the original RUCM template and restriction rules [11]. Following [9], security use cases in RMCM specify flow of events mitigating specific steps in misuse cases. But, to reduce ambiguity and produce a structured, analyzable specification, we propose to use RUCM for eliciting security use cases as we do for use cases, with the only addition of a field called 'Compliance' to specify the standard provisions that the security use case should comply with (see Fig. 7). For the details of the RUCM template, the reader is referred to [11].

Fig. 7 shows a simplified security use case (only some fields are shown) for mitigating the threat *Get Unauthorized Access via SQLi*. It provides compliance (Line 3) with a clause in the widely-used security standard — ISO/IEC 27001:2013 Information Security Management Systems Requirements.

1	**SECURITY USE CASE** Validate Website Inputs
2	**Precondition** The system has received some inputs.
3	**Compliance** ISO/IEC 27001:2013 clause A.9.4:System and application access control.
4	**Basic Flow**
5	1. The system sanitizes the inputs according to the input specification.
6	2. The system VALIDATES THAT the inputs are valid.
7	**Postcondition** The system has successfully validated the inputs.
8	**Specific Alternative Flow**
9	RFS 2
10	1. The system displays an error message.
11	2. ABORT.

Fig. 7. A security use case specification

C. Mitigation Schemes

To address *Challenge 3*, RMCM provides the mitigation template given in Table III. The field 'Mitigation Scheme' in misuse case specifications refers to the scheme mitigating misuse cases (Table I). Mitigation schemes themselves are specified in a separate table, according to the mitigation template, to facilitate reuse. Differently from security use cases specifying flow of events mitigating a specific threat scenario, mitigation schemes provide the secure coding methods adopted by the system, guidelines on how to educate users and other mechanisms to prevent various security threats in general. As different security threats can be mitigated by applying standard secure coding methods, such as those listed in OWASP [23], once a mitigation scheme is specified, it can be reused or tailored as necessary for various security threats. The field 'Mitigated Misuse Case' in Table III lists such various security threats mitigated by a given scheme, while the field 'Compliance' lists the standard provisions that the mitigation scheme addresses.

The mitigation schemes complement the security use cases. The field 'Compliance' in these two artifacts provides precise traceability to specific clauses in standard provisions, and together these artifacts provide a means for stakeholders to demonstrate compliance with applicable security and privacy standards and regulations. For instance, the mitigation scheme in Fig. 8 mitigates two misuse cases — *Expose Information via Insecure Data Storage* and *Expose Information due to Insecure Authentication*. It also supports compliance with some of the clauses in ISO/IEC 27001:2013. On our website [17], we give two additional mitigation schemes which are used to mitigate various security threats for EDLAH2.

TABLE III
MITIGATION TEMPLATE

Scheme Name	The name of the mitigation scheme.
Brief Description	A short description about the mitigation scheme.
Actor	The actor who is responsible for reviewing and/or implementing the mitigation tasks.
Mitigated Misuse Case	Mitigate relationships to the misuse case(s). It specifies the misuse case(s) mitigated by the mitigation scheme.
Compliance	Specifies the standard/applicable provision(s) that this mitigation scheme provides compliance.
Mitigation Tasks	Specifies the mitigation tasks. Tasks(numbered) Mitigation Tasks

VI. Tool Support

We implemented a tool, *RMCM-V* for checking the consistency between the misuse case diagram and the specifications, and between the specifications and the RMCM template.

Scheme Name	Secure Coding for Mobile Program.	
Brief Description	This mitigation scheme mitigates serious and common security threats for mobile apps.	
Actor	Software Developer, Security Engineer.	
Mitigated Misuse Case	Expose Information via Insecure Data Storage, Expose Information due to Insecure Authentication.	
Compliance	ISO/IEC 27001:2013 clause A.6.1.5:Information security in project management, clause A.9.4:System & application access control, clause A.10.1:Cryptographic controls.	
Mitigation Tasks	1	OBFUSCATE all apk files using an Android apk obfuscator.
	2	ENCRYPT sensitive data stored in mobile device, such as SQLite database, cache and log files, and SD card.
	3	Apply root detection check. If jailbreak is detected, the system warns the client of privacy data leakage risk.
	4	Periodically clear caching data automatically.
	5	Do not grant files world readable or writable permissions.
	6	Perform code integrity violation check.
	7	Educate users not to download apps from unofficial stores.

Fig. 8. A sample mitigation scheme

RMCM-V automatically checks consistency and reports inconsistencies such as a misuse case diagram missing a *threaten* or *mitigate* relationship in specifications. The tool architecture is composed of three layers (see Fig. 9): (i) *the User Interface (UI) layer*, (ii) *the Application layer*, and (iii) *the Data layer*.

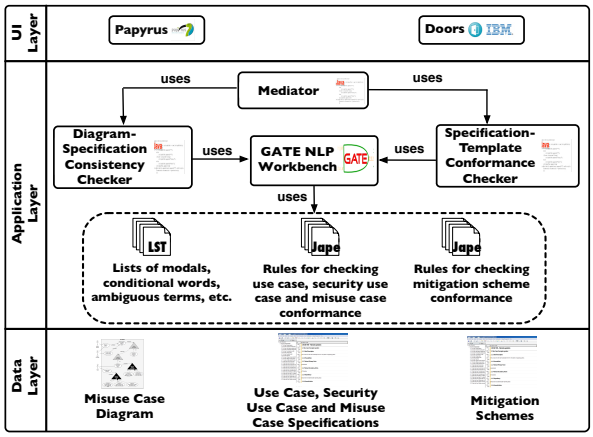

Fig. 9. Layered architecture of RMCM-V

User Interface (UI) Layer. This layer supports the activities of eliciting security and (mis)use cases, and mitigation schemes (see Fig. 4). We employ IBM Doors [19] for eliciting security and (mis)use case specifications and mitigation schemes according to the RUCM and RMCM templates and their restriction rules. We employ Papyrus [18] for misuse case diagrams. Sindre and Opdahl [9] proposed a metamodel of the basic misuse case concepts and their relation to the UML metamodel. We adopted and implemented their proposed metamodel as a UML profile in Papyrus so that we can use the Papyrus model editor for drawing misuse case diagrams.

Application Layer. This layer supports the main activities of our modeling method in Fig. 4: *checking conformance for diagram and specifications* and *checking conformance for mitigation schemes*. It contains three main components implemented in Java: *Mediator*, *Diagram-Specification Consistency Checker*, and *Specification-Template Conformance Checker*. To access these *Application Layer* components through the *UI Layer*, we implemented an IBM Doors plugin.

The *Mediator* is a coordinator that manages the other two components. The *Specification-Template Conformance*

Checker employs NLP to check whether the specifications and mitigation schemes comply with the RUCM and RMCM templates and their restriction rules. NLP is also used by the *Diagram-Specification Consistency Checker* to check the consistency between the misuse case diagram and specifications.

To support NLP, we employ a regular expression engine, called JAPE [49], in the GATE workbench (http://gate.ac.uk/), an open-source NLP framework. We implemented the restriction rules in JAPE. First, the specifications are split into tokens. Second, Part-Of-Speech (POS) tags (i.e., *verb*, *noun*, and *pronoun*) are assigned to each token. By using the restriction rules implemented in JAPE, blocks of tokens are tagged to distinguish RUCM/RMCM steps (e.g., *actor to system interaction*, *malicious actor to system interaction*, and *internal actions*), types of flows (i.e., *threat-specific*, *alternative*, and *global*), and mitigation scheme tasks. The NLP output contains the annotated use case steps and mitigation scheme tasks. The *Diagram-Specification Consistency Checker* and *Specification-Template Conformance Checker* process these annotations with the misuse case diagram to generate the list of inconsistencies among artifacts.

Data Layer. The specifications and the mitigation schemes are stored as native IBM Doors format. The misuse case diagram is stored using the UML profile mechanism.

VII. Industrial Case Study

We applied RMCM to the security and privacy requirements of EDLAH2. Our goal was to assess, in an industrial context, how RMCM can improve the practice of eliciting and analyzing security and privacy requirements and how well RMCM addresses the challenges that we identified in Section II. EDLAH2 engineers provided their initial EDLAH2 documentation, which contained a use case diagram and use case specifications. To model the security and privacy requirements of EDLAH2 according to RMCM, we first examined the initial EDLAH2 documentation and then worked with EDLAH2 engineers to build and iteratively refine our models for security and privacy requirements. Tables IV reports the size of the resulting RMCM artifacts. Table V reports the number of times each restriction rule (*R1-R15* in Table II) is applied when eliciting misuse cases.

TABLE IV
The size of the RMCM artifacts in EDLAH2

	no.	relations	alt. flows	alt. threat flows	steps	malicious steps
Use cases	9	9	25	NA	143	NA
Security use cases	4	26	4	NA	12	NA
Mitigation schemes	3	20	NA	NA	NA	NA
Misuse cases	17	17	21	10	145	22

In Table IV, the column 'no.' shows the numbers of use cases, security use cases, mitigation schemes, and misuse cases we modeled. The column 'relations' shows the numbers of *include* (second row), *mitigate* (third and fourth rows), and *threaten* (fifth row) relations among those artifacts. The columns 'alt. flows', 'alt. threat flows', 'steps', 'malicious steps' show the numbers of alternative flows, alternative threat flows, steps, and malicious steps, respectively. 'Malicious steps' denotes the steps in misuse case specifications that correspond to interactions between malicious actors and the system. 'NA' denotes "not applicable".

TABLE V
Number of Appearances of RMCM Restrictions in EDLAH2

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
80	38	2	0	0	0	4	1	1	1	1	9	2	14	3

As shown in Table IV, we elicited 17 misuse cases threatening 9 use cases. We explicitly captured 10 alternative threat flows (column 'alt. threat flows'), specifying the conditions and the flow of events that may still lead to successful attacks when the attacks specified in the basic threat flows fail. We elicited 21 scenarios that do not pose security threats (column 'alt. flows'). We also elicited 4 security use cases and 3 mitigation schemes. They typically mitigate more than one misuse case since there are 26 *mitigate* relations between security use cases and misuse cases and 20 *mitigate* relations between mitigation schemes and misuse cases (column 'relations').

One first interesting observation is that usually several potential threats are relevant for each use case. This makes security requirements engineering rather complex, especially when involving many stakeholders, and it is therefore highly important to be systematic and structured in identifying and specifying security requirements. A second observation is that a given threat can materialize through multiple threat scenarios, which need to be all identified and carefully analyzed. It is therefore important to have a structured and precise way to express such scenarios. Our case study also confirmed that mitigation schemes can be reused across multiple misuse cases and they are therefore useful reusable artifacts that should be captured independently. One other interesting result is that we also identified 22 malicious steps, which contain information about the attack surfaces, i.e., parameters, URLs, files, and programs, through which the security attacks can be carried out. Such malicious steps could be precisely identified because of the systematic use of restriction rules. With so many malicious steps, the resulting attack surface is rather complex and justifies the need to be rigorous and systematic. Malicious steps are therefore expected to provide insightful information to security testers of EDLAH2 apps.

Thanks to the RMCM extensions (Table II) that reflect common security threats for mobile apps, we were able to cover security threats relevant to EDLAH2 in a systematic and precise way. It is interesting to note that some of them were not covered in the initial EDLAH2 documentation. For instance, the extensions helped us identify and model that *getting unauthorized access* is an SQLI injection attack (*R3* and *R12*) while *exposing information from mobile* is a security threat that exploits insecure data storage (*R2* and *R11*), which was not previously documented. As shown in Table V, we applied almost all the proposed restriction rules to systematically model the security threats. Only three restriction rules (*R4, R5, R6*) were not used since they correspond to security threats targeting XML and LDAP databases, and EDLAH2 only uses an SQL database.

To evaluate the RMCM output in light of the challenges we identified earlier, we had semi-structured interviews with four engineers holding various roles in EDLAH2 consortium (i.e., project manager, software engineer, and game architect). They all had substantial software development experience, ranging from 3 to 28 years. All participants had experience with use case driven development and modeling. The interview included a presentation illustrating the RMCM steps, a tool demo, and examples from EDLAH2. To capture the perception of engineers participating in the interview, regarding the potential benefits of RMCM, and assess the extent to which it addresses the targeted challenges, we handed out a questionnaire [50] including questions to be answered according to a Likert scale [51] (i.e., strongly agree, agree, disagree, and strongly disagree) along with open, written comments. It was structured for the participants to assess RMCM in terms of adoption effort, expressiveness, and comparison with current practice.

Results from the interview showed that all participants agreed on the following positive aspects of RMCM:

- The security extensions in RMCM provided enough expressiveness to conveniently capture security and privacy requirements in EDLAH2.
- RMCM provided useful assistance for systematically capturing and analyzing security and privacy requirements, especially when compared to the current, informal practice in EDLAH2.
- Mitigation schemes and security use cases enabled precise traceability to specific clauses in the security standard provisions of ISO/IEC 27001:2013, which EDLAH2 aims to comply with, as illustrated in Fig. 7 and Fig. 8.
- RMCM-V provided useful assistance for minimizing inconsistencies in the RMCM artifacts of EDLAH2. It also helped ensure that the restrictions in Table II were accurately followed in the misuse case specifications.

We, together with the participants, also identified a number of challenges regarding the application of RMCM:

- *Adopting RMCM in an organization*. In the current practice in EDLAH2, like in many other environments, there is no systematic way to capture security requirements in use case models. Even though the effort required to apply our modeling approach was considered to be reasonable by EDLAH2 engineers, it may be a challenge to convince engineers in general to engage in this additional modeling effort. The costs and benefits of such an activity should be further evaluated to help with adoption. This is, however, a common and general challenge when introducing new practices in software development.
- *Additional extensions in RMCM*. The security extensions in RMCM cover various security and privacy concerns to be captured in use case models. However, due to rapidly changing software and hardware technology, new types of security threats will likely need to be covered with further security extensions. In a way, such extensions can be treated as a knowledge repository of potential vulnerabilities and their associated mitigation schemes, that has to be regularly updated and helps creating awareness of security threats and solutions across an organization.

Our discussion with the participants resulted in the following priorities for future work:

- *Providing more automation*. Although RMCM-V provides useful assistance for minimizing inconsistencies among RMCM artifacts, more automation, such as keyword completion and semi-automated creation of alternative flows, would help engineers when writing misuse case and security use case specifications.
- *Generating security test cases*. Restricted misuse case specifications contain the precise control flow corresponding to various threat scenarios in an analyzable form. Such information should be used to automate test generation for requirements-driven security testing.

Threats to validity. The main threat to the validity of our case study regards the generalizability of the conclusions. To mitigate the threat, we applied RMCM to a representative mobile application that includes nontrivial use cases in an application domain entailing numerous and varied security threats. We selected the respondents to our interviews to hold various roles and with substantial industry experience. To limit threats to the internal validity of the case study, we had many meetings with the engineers in EDLAH2 to verify the correctness and completeness of our models.

VIII. Conclusion

This paper presents a use case-driven security requirements modeling method, called RMCM, for documenting the security and privacy requirements of mobile apps in a structured and analyzable form. Our main motivation is to enable security and privacy requirements modeling by relying on commonly used artifacts in use-case driven development and by adding a limited number of extensions, thus achieving widespread applicability. RMCM builds on and integrates existing work and is supported by a tool employing natural language processing for checking the consistency of artifacts and compliance to templates. The key characteristic of our method is that it captures threat scenarios and mitigation schemes in an explicit and structured form, thus enabling both automated analysis of threat scenarios, e.g., consistency and conformance checking, and reuse of mitigation schemes. Initial results from structured interviews with experienced engineers suggest that RMCM is accurate and practical to capture security and privacy requirements of mobile apps in industrial settings.

In addition to supporting more precise and complete security requirements, RMCM is a first step to achieve a longer-term objective: automated test generation for requirements-driven security testing. Our plan for the next stages is to provide an automated technique that generates security test cases from misuse case models. Our ultimate objective is to achieve adequate coverage of the specified security and privacy requirements, with traceability information between security requirements and generated test cases.

Acknowledgment. This work is supported by the National Research Fund, Luxembourg FNR/P10/03, INTER/AAL/15/11213850, and INTER/DFG/14/11092585.

REFERENCES

[1] A. K. Jain and D. Shanbhag, "Addressing security and privacy risks in mobile applications," *IT Professional*, vol. 14, no. 5, pp. 28–33, 2012.

[2] "EDLAH2: Active and Assisted Living Programme," http://www.aal-europe.eu/projects/edlah2/.

[3] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," in *MindTrek'11*. ACM, 2011, pp. 9–15.

[4] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *ACSAC'99*, 1999.

[5] J. McDermott, "Abuse-case-based assurance arguments," in *ACSAC'01*, 2001.

[6] D. G. Firesmith, "Security use cases," *Journal of Object Technology*, vol. 2, no. 3, pp. 53–64, 2003.

[7] A. L. Opdahl and G. Sindre, "Experimental comparison of attack trees and misuse cases for security threat identification," *Information and Software Technology*, vol. 51, pp. 916–932, 2009.

[8] L. Rostad, "An extended misuse case notation: Including vulnerabilities and the insider threat," in *REFSQ'06*, 2006, pp. 33–43.

[9] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering*, vol. 10, pp. 34–44, 2005.

[10] ——, "Templates for misuse case description," in *REFSQ'01*, 2001.

[11] T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, pp. 1–38, 2013.

[12] C. Wang, F. Pastore, A. Goknil, L. C. Briand, and M. Z. Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *ISSTA'15*, 2015, pp. 385–396.

[13] ——, "UMTG: a toolset to automatically generate system test cases from use case specifications," in *ESEC/FSE'15*, 2015, pp. 942–945.

[14] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Applying product line use case modeling in an industrial automotive embedded system: Lessons learned and a refined approach," in *MODELS'15*, 2015, pp. 338–347.

[15] ——, "Configuring use case models in product families," *Software and Systems Modeling*, 2016.

[16] ——, "PUMConf: a tool to configure product specific use case and domain models in a product line," in *FSE'16*, 2016, pp. 1008–1012.

[17] X. P. Mai, "RMCM-V: a tool for checking consistencies between misuse case diagram, specifications, and restricted misuse case modeling templates," https://sites.google.com/site/rmcmverifier/, 2017.

[18] "Papyrus," https://www.eclipse.org/papyrus.

[19] "IBM Doors," http://www.ibm.com/software/products/ca/en/ratidoor.

[20] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2001.

[21] F. Armour and G. Miller, *Advanced Use Case Modeling: Software Systems*. Addison-Wesley, 2001.

[22] D. Kulak and E. Guiney, *Use Cases: Requirements in Context*. Addison-Wesley, 2003.

[23] "OWASP Top 10 Mobile Security Risks," https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10.

[24] B. Fabian, S. Gurses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements Engineering*, vol. 15, pp. 7–40, 2010.

[25] D. Mellado, C. Blanco, L. E. Sanchez, and E. Fernandez-Medina, "A systematic review of security requirements engineering," *Computer Standards & Interfaces*, vol. 32, pp. 153–165, 2010.

[26] A. Souag, R. Mazo, C. Salinesi, and I. Comny-Wattiau, "Reusable knowledge in security requirements engineering: a systematic mapping study," *Requirements Engineering*, vol. 21, pp. 251–283, 2016.

[27] P. Salini and S. Kanmani, "Survey and analysis on security requirements engineering," *Computers and Electrical Engineering*, vol. 38, pp. 1785–1797, 2012.

[28] I. A. Tondel, M. G. Jaatun, and P. H. Meland, "Security requirements for the rest of us: A survey," *IEEE Software*, vol. 25, no. 1, pp. 20–27, 2008.

[29] S. Gurses, B. Berendt, and T. Santen, "Multilateral security requirements analysis for preserving privacy in ubiquitous environments," in *UKDU'06*, 2006.

[30] N. R. Mead, E. D. Hough, and T. R. Stehney, "Security quality requirements engineering (square) methodology," Carnegie Mellon Software Engineering Institute, CMU/SEI-2005-TR-009, 2005.

[31] T. Lodderstedt, D. A. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *UML'02*, 2002, pp. 426–441.

[32] L. Liu, E. Yu, and J. Mylopoulos, "Security and privacy requirements analysis within a social setting," in *RE'03*, 2003, pp. 151–161.

[33] H. Mouratidis and P. Giorgini, "Secure tropos: a security-oriented extension of the tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 2, pp. 285–309, 2007.

[34] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *RE'04*, 2004, pp. 354–355.

[35] D. Hatebur, M. Heisel, and H. Schmidt, "Security engineering using problem frames," in *ETRICS'06*, 2006, pp. 238–253.

[36] F. den Braber, I. Hogganvik, M. S. Lund, K. Stolen, and F. Vraalsen, "Model-based security analysis in seven steps — a guided tour to the CORAS method," *BT Technology Journal*, pp. 101–117, 2007.

[37] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone, "From trust to dependability through risk analysis," in *ARES'07*, 2007, pp. 19–26.

[38] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A comparison of the Common Criteria with proposals of information systems security requirements," in *ARES'06*, 2006, pp. 654–661.

[39] ——, "Applying a security requirements engineering process," in *ESORICS'06*, 2006, pp. 192–206.

[40] I. Alexander, "Misuse cases: Use cases with hostile intent," *IEEE Software*, vol. 20, no. 1, pp. 58–66, 2003.

[41] ——, "Misuse cases help to elicit non-functional requirements," *Computing & Control Engineering Journal*, vol. 14, no. 1, pp. 40–45, 2003.

[42] ——, "Initial industrial experience of misuse cases in trade-off analysis," in *RE'02*, 2002, pp. 61–70.

[43] D. G. Rosado, E. Fernandez-Medina, and J. Lopez, "Applying a UML extension to build use cases diagrams in a secure mobile grid application," in *ER'09 Workshops*, 2009, pp. 126–136.

[44] G. Sindre, A. L. Opdahl, and G. F. Brevik, "Generalization/specialization as a structuring mechanism for misuse cases," in *SREIS'02*, 2002.

[45] G. Sindre, D. G. Firesmith, and A. L. Opdahl, "A reuse-based approach to determining security requirements," in *REFSQ'03*, 2003.

[46] M. El-Attar, "Towards developing consistent misuse case models," *Journal of Systems and Software*, vol. 85, no. 2, pp. 323–339, 2012.

[47] ——, "Using SMCD to reduce inconsistencies in misuse case models: A subject-based empirical evaluation," *Journal of Systems and Software*, vol. 87, pp. 104–118, 2014.

[48] J. Whittle, D. Wijesekera, and M. Hartong, "Executable misuse cases for modeling security concerns," in *ICSE'08*, 2008, pp. 121–130.

[49] H. Cunningham et al, "Developing language processing components with gate version 8 (a user guide), http://gate.ac.uk/sale/tao/tao.pdf."

[50] [Online]. Available: https://goo.gl/forms/OrAZcZLvsVm5tUN13

[51] A. N. Oppenheim, *Questionnaire Design, Interviewing and Attitude Measurement*. Continuum, 2005.