SeMA: A Design Methodology for Building Secure Android Apps

Joydeep Mitra           Venkatesh-Prasad Ranganath
Kansas State University, U.S.A
{joydeep,rvprasad}@k-state.edu


Abstract
UX (user experience) designers visually capture the UX of an app via storyboards. This method is also used in Android app development to conceptualize and design apps.

Recently, security has become an integral part of Android app UX because mobile apps are used to perform critical activities such as banking, communication, and health. Therefore, securing user information is imperative in mobile apps.

In this context, storyboarding tools offer limited capabilities to capture and reason about security requirements of an app. Consequently, security cannot be baked into the app at design time. Hence, vulnerabilities stemming from design flaws can often occur in apps. To address this concern, in this paper, we propose a storyboard based design methodology to enable the specification and verification of security properties of an Android app at design time.

1 Why a new methodology?

Android app development teams use storyboarding as part of their design process [21, 14]. A storyboard is a sequence of images that serves as a specification of the user observed behavior in terms of screens and transitions between screens. Storyboarding helps designers identify different kinds of app users (user profiles), explore possible real world scenarios in which a user will interact with the app, and develop wireframes to capture the scenarios for identified user profiles [14, 9, 12]. Numerous tools such as Xcode [1], Sketch [18], OmniGraffle [11], and Android JetPack’s Navigation Component [6] help designers digitally express the storyboard of an app.

Storyboading is meant to be participatory because it can be used by designers to get feedback from potential users about likely user scenarios, and from developers about the technical challenges of implementing the design. However, in their current form, storyboards do not capture non-functional requirements such as security. Consequently, Android app designers and developers cannot use storyboarding to collaborate and express security requirements at design time. As a result, verification of security requirements is delayed until later stages. This delay increases cost of app development as vulnerabilities due to flaws detected in later stages increase the cost of development [20].

In this context, to enable reasoning about security at design time, we are exploring two ideas: 1) extend storyboards to capture non-functional properties and 2) develop a methodology that uses storyboards to specify and verify security requirements of apps at design time.

To understand the reasons for and benefits of these ideas, we need to understand the current landscape of Android app development practices.

Today, developers consider the security requirements of an app while or after implementing the app. Existing research efforts related to Android app security have focused on developing tools and techniques to help detect vulnerabilities in an app’s implementation (as opposed to in an app’s design) [19, 8]. Despite such efforts, apps with known vulnerabilities find their way to app stores [16, 4]. This is (possibly) because existing tools are neither accurate [15] nor scalable [13] in terms of detecting known vulnerabilities. A complementary approach to help secure the Android ecosystem is to develop tools and techniques to identify malicious apps and keep them out of the ecosystem. However, the lack of such tools that accurately (i.e., low false-positives and false-negatives) identify malicious apps and the innocent-until-proven-guilty philosophy
adopted by most Android app stores (e.g., Google Play) enables malicious apps to enter app stores \[7, 22\]. So, if users install malicious apps and vulnerable apps exist, then the malicious apps can exploit the vulnerable apps to cause harm to the user. Given this landscape, we believe it is worthwhile to explore the unexplored: preemptively eliminate vulnerabilities in apps by enabling developers and designers to consider security at the design stage of app development process so that malicious apps cannot exploit the apps.

Besides complementing existing techniques, such exploration will help consider the following research questions:

a) How much effort is required in terms of time and cost to formally reason about Android apps at the design level?
b) In the context of Android app development, is secure-by-design cost effective?
c) Is verifying an app based on its storyboard easier than verifying it based on its code?
d) In terms of improving security, how will the proposed approach compare to existing curative approaches, i.e., detect vulnerabilities after they occur?

2 What is the methodology?

Motivated by the above reasons, we propose the following methodology, Securing Mobile Apps (SeMA), that borrows heavily from model driven development.

1) App development begins with the storyboard (model) of an app that captures the screens of the app, the transitions between screens, and the resources used by the app.

2) The storyboard is iteratively refined by adding behavioral and security properties.

3) As the storyboard is refined, verification techniques are used to check the behavior of the storyboard satisfies the given security properties.

4) Once the refined storyboard satisfies the desired requirements, property preserving structural code of the app is automatically generated.

5) Business logic is added to the generated code.

2.1 An Illustrative Example

To understand SeMA, consider the development of an app that sends a message to a set of pre-identified contacts at the push of a button.

2.1.1 App Specification

Figure 1 shows the storyboard of this example app along with a brief description of various concepts captured in the storyboard.

Initially, the app designer specifies the required screens and the transitions between the screens as shown in the top left corner of the figure (labeled A). The initial snapshot is similar to how storyboards are used in designing apps today. Besides the graphical description of screens of the app, this snapshot captures the navigational possibilities in the app: 1) user will see the Messenger screen up on launching the app, 2) the user can transition to either Contacts or MsgStatus screens from the Messenger screen, and 3) the user can transition to SaveStatus screen from the Contacts screen.

Next, the app designer refines the initial snapshot with constraints based on user actions (marked in red). For example, when the user clicks the Add button in the Messenger screen, the app transitions to the Contacts screen.

As the next refinement, the app developer (in collaboration with the app designer) adds more behavioral constraints to the storyboard. For example, a constraint based on the state of the app (marked in blue) is
Figure 1: Storyboard of the Emergency App. The clouds are not part of the storyboard; they provide information about the different graphical elements in the figure.

added to the transition from the Contacts screen. Other transitions in the app are enriched with constraints in subsequent refinements.

Many constraints rely on operations involving the resources in the device. While such resources are pre-defined, apps can specify views of resources that they will use.

The resource view of the app (shown in Figure 1) describes the view of the resources used in the example app as follows.

- **EXT_STORE** denotes the external storage on Android devices that can be accessed by all apps installed on the device. The app uses the following two capabilities of this resource: write(f,p) writes the value p into file f and read(f) reads a value from file f and returns it.

- **SMS** denotes a pre-defined service available on Android devices that the app uses to send a SMS message m to a list of phone number p via send(m,p) operation.

The final storyboard with all the behavior and resource operations specified will look like Figure 1.

To give an example of how the constraints in a transition are interpreted, the transition from the Messenger screen to the MsgStatus screen is taken when 1) the SendMsg button is pressed and 2) an SMS message is successfully sent to the phone numbers in the file named MyContacts.txt.
2.1.2 Analysis

In this example, suppose we are interested in the violation of this property: *malicious input should not influence data flow in the app.*

Based on the semantics of \texttt{EXT\_STORE}, all apps can access data in \texttt{EXT\_STORE}. Hence, \texttt{EXT\_STORE} can possibly contain malicious inputs as malicious apps can modify the contacts stored by the example app. Since the phone numbers from \texttt{EXT\_STORE} is used in \texttt{SMS.send(...)}, a violation of the property is detected by analyzing the data flow through the storyboard in accordance with the semantics of resources used by the app.

This vulnerability can be fixed by storing the phone numbers in a more secure way. For example, \texttt{MyContacts.txt} file can be stored in the app’s internal storage that is accessible only by the app. This fix can be suggested by tooling support, expressed by the developer in the storyboard by using appropriate resources, and enforced via code generation.

3 Design Choices

3.1 Support visual-friendly specification

Realizations of SeMA will require a specification language in which the behavioral and security properties can be specified. Since the proposed methodology is based on storyboarding, the specification language will have a visual aspect, i.e., to work with UI elements. In the domain of Android app development, a visual specification language can have numerous advantages over textual specification languages: (1) A visual specification based on storyboards is closer to the developer’s and designer’s mental model of an app, i.e., a graph of screens; and (2) A visual specification will preserve the participatory nature of storyboarding by allowing designers and developers to collaborate on a common substrate – storyboard. Prior efforts suggest that graphical representation of complex algorithms are sometimes easier to comprehend than textual representations [17]. In a similar vein, we believe visual representation of an app’s behavior and security requirements will ease the understanding of app security and result in secure apps.

3.2 Leverage existing mobile app design frameworks

Mobile app developers are already using frameworks and libraries such as the Android JetPack Navigation component [6] to design their apps in the form of navigation graphs containing screens, transitions between screens, and user actions that trigger the transitions. However, they do not support the specification of richer behavior needed to perform analysis and verification of security properties. We plan to realize SeMA on top of such existing support to ease the adoption and use of SeMA.

3.3 Support pre-defined security policies

We do not want to burden app developers or designers with specifying the security policies required for verification because specifying such policies may initially be a non-trivial burden for the average app developer or designer. Moreover, requiring app developers or designers to specify such policies will unnecessarily complicate the existing design process and raise the bar for adopting SeMA. Therefore, we envision SeMA will initially support the checking of pre-defined security properties that are based on classes of common vulnerabilities [10] as illustrated in Section 2.1.2.

Existing efforts have identified vulnerabilities that commonly plague Android apps and have developed benchmarks to capture them [2]. Ghera is one such benchmark suite that captures 60 known vulnerabilities in Android apps [10]. Most of the vulnerabilities captured in Ghera can be classified as follows:

1. **Reliance on Data from Potentially Malicious Sources**: An Android app X can interact with other apps. Consequently, other apps can request an app X to perform an action on their behalf. If app X blindly trusts an input from an external source (a component not in app X) and uses it to manipulate data or make a critical decision, then app X is vulnerable to exploitation.
2. **Disclosure of Sensitive Information**: An app can accidentally expose sensitive information. For example, suppose an app needs device ID to track the devices it is installed on. For this purpose, it accesses device ID (after asking permission from the user) and saves it accidentally in external storage. Since external storage can be accessed by all apps in the device, a malicious app can access the device ID without having appropriate permissions.

3. **Exposure of Privileged Resources**: Android apps can access privileged resources only if the apps are granted required permission by the user or the Android system. An app with access to privileged resources can accidentally enable privilege escalation. For example, suppose app X has been granted permission to read the user’s SMS. If app X is designed with a component that reads user messages and can be accessed by other apps, then a malicious app can retrieve user messages even without having required permissions.

The above classes of vulnerabilities stem from design oversight and SeMA is well-suited to eliminate them.

However, there are vulnerabilities that stem from implementation errors, e.g., using the Block Cipher ECB Algorithm to encrypt sensitive information or storing the encryption key in source code [3]. While such vulnerabilities cannot be eliminated by design verification, they can be eliminated by using smart code generation schemes.

### 4 Challenges

To successfully realize the proposed methodology, the following interesting challenges need to be addressed while developing SeMA.

1. **Storyboard extensions to capture non-UI behavior**: Today, storyboards of Android apps capture only the UI components of an app and behavior triggered by user actions, e.g., user action triggered transitions between screens. However, Android apps also exhibit non-UI related behavior, e.g., reading a file from the app’s storage as shown in the earlier illustrative example. Therefore, existing storyboards will have to be extended with features to capture behavior that stem from non-UI elements without disrupting the workflow and the mental model associated with storyboards.

2. **Context-aware analysis**: Android apps do not run in isolation. They communicate with remote servers, other apps on the device, the underlying Android framework, etc. Therefore, the security of an app depends on the context in which it is operating. Security analysis of app storyboards will have to be aware of such contexts; otherwise, the analysis will be imprecise.

3. **Reactive nature of Android apps**: Android apps interact with the underlying platform via platform-defined application lifecycle methods to handle the numerous system and user events. These events could trigger the execution of an app from different entry points. Also, apps have the ability to persist information across different executions. This allows for different executions of an app that start at different entry points to be implicitly related by persisted data. This possibility should be considered to accurately reconstruct the behavior of an app for the purpose of security reasoning.

4. **Scalability of analysis**: To be accurate and effective, security analysis of Android apps will have to consider various contexts and rich features of the Java (or Kotlin) programming language used to implement the apps. Such analysis are neither scalable [13] nor accurate [15] as they are plagued by the challenges faced by non-trivial source code analysis [8]. However, similar analysis can be performed easily at design time on abstract models such as storyboards that involve finite number of screens and transitions and possibly smaller value domains. However, even such design-time analysis can face scalability issues as the richness of both the security properties to be checked and the app behavior captured in storyboards increases.

5. **Evaluation of SeMA**: Evaluating the effectiveness of SeMA will involve conducting a usability study. Designing an effective usability study has numerous challenges [5] such as getting users to use the methodology, selecting a representative group of users, developing the metrics to measure usability etc.
References


