Automatically Inferring and Enforcing User Expectations

Jenny Hotzkow
Saarland University
Saarbrücken, Germany

ABSTRACT

Can we automatically learn how users expect an application to behave? Yes, if we consider an application *from the users perspective*. Whenever presented with an unfamiliar app, the user not only regards the context presented by this particular application, but rather considers previous experiences from other applications. This research presents an approach to reflect this procedure by automatically learning user expectations from the semantic contexts over multiple applications. Once the user expectations are established, this knowledge can be used as an oracle, to test if an application follows the user's expectations or entails surprising behavior by error or deliberately.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; • **Software and its engineering → Model checking**; Dynamic analysis; Automated static analysis;

KEYWORDS

android, anomaly detection, malware classification, semantics-aware

ACM Reference format:
Jenny Hotzkow. 2017. Automatically Inferring and Enforcing User Expectations. In *Proceedings of 26th ACM SIGSOFT International Symposium on Software Testing and Analysis , Santa Barbara, CA, USA, July 2017 (ISSTA'17-DOC),* 4 pages.
https://doi.org/10.1145/3092703.3098236

1 INTRODUCTION

The Android market of mobile applications is rapidly growing and the amount of malicious software is evermore increasing [9]. This motivated a large research base in malware classifiers [1, 5, 8] and tools aiming to identify and reduce the possible attack surface by removing over-privileges. Still, as this field of research evolves, so do malicious applications. Therefore, the user should be able to decide *what risk she is willing to take for which functionalities.* But even as the Android permission and enforcement systems gets improved and extended [3], users are lacking the necessary insight to understand the impact of permissions and are not able to specify the mostly complicated policies on their own.

ISSTA'17-DOC, July 2017, Santa Barbara, CA, USA

ACM ISBN 978-1-4503-5076-1/17/07…$15.00
https://doi.org/10.1145/3092703.3098236

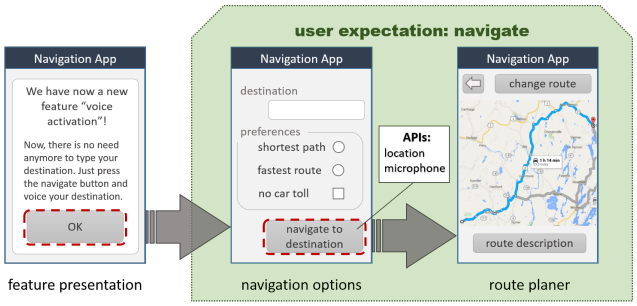

Figure 1: Navigation example: the text below each screen describes the semantic topic from a users perspective.

For a navigation app, the user would intuitively expect that the application accesses the current location and even the file system for off-line stored maps. Meanwhile, an access to the microphone, which does not affect any visible functionality, is rather surprising and indicates suspicious behavior. Existing approaches, are able to predict on the application level if certain behavior is reasonable. For example CHABADA [5] would be able to identify if the app description indicates any microphone usage. But the application layer is not fine-grained enough as it is not apparent when in the execution and under which conditions specific behavior is triggered.

Consider the example in Figure 1. The user is presented with a screen announcing a feature 'voice activation' and this feature is used on the next screen. The user is able to relate the semantic context of the feature presentation to the later microphone usage. Therefore, the user would likely consider the usage of the microphone as expected behavior. Meanwhile, a microphone activation without this information would be rather surprising. Moreover, an app description could mention a voice activation feature which is not accessible. Meaning, the microphone usage has no visible effect. This indicates either an error or hidden behavior.

In this work, I want to learn user expectations on user interface (UI) level, i.e. from UI elements. A *user expectation* entails the desire to access a certain functionality and the expectation of underlying behavior. For example, a user who aims to navigate to another location, concludes from the UI that the 'navigate to destination' button will provide a route by use of the current position. As evident from the example in Figure 1, it is necessary to analyze the semantic context of the interacted element, current screen and screens presented before the action. *The application behavior should be reasonable from the UI context.* Consequently, this gives an estimation on how far the actual app behavior complies with the user expectations.

The hypothesis that *a user is already familiar with other applications which may share similar functionalities*, like alternative navigation applications, is intuitive. The knowledge from these apps complements the expectations on the particular application. Accordingly, user expectations can be analyzed over multiple applications to identify *universal user expectations* which represent

the *common expectations for similar functionalities*, also referred to as *features*. The user expectation for 'navigate' should be common among all navigation applications.
The proposed research not only allows to automatically generate user expectations, but can be utilized

- to extract *user expectation based rules*, with the necessary insight of the security impact on accessible features
- to create an *oracle* for test generation—by transferring user expectations to execution traces, test-cases can be generated, which allow to detect if user expectations are correctly achieved;
- to identify *surprising* behavior, i.e. behavior that is not backed up by the displayed UI content, caused by errors or deliberately by malicious applications;
- to serve as a *test coverage* criterion, specifying how many user expectations related to different features were covered

For this purpose, I will investigate the following research questions:

(RQ1) To which extent can user expectations be expressed using a *behavioral model*—an application state model based on the sequence of UI screens, including semantic context and the respective UI interactions with the related privileges?
(RQ2) Are there nontrivial universal expectations for app categories which are applicable across applications?
(RQ3) Can universal expectations be used to generate test cases and serve as an oracle?
(RQ4) Can user expectations be used to detect surprising behavior?

To answer these questions, I plan to use static and dynamic analysis to extend state-of-the-art behavioral models [4] with semantic context information, for each element and the overall screen. This novel model is used to extract user expectations based on the UI content, for the currently presented screen and for a sequence of consecutive screens. These expectations can be mined over multiple applications to identify patterns which reflect universal expectations and can be linked to specific features or application categories.

2 BACKGROUND

Privileges. To gain access to security sensitive data or hardware components, an application has to be granted the respective system resource access. In Android this is implemented with a permission system, which requires the users to agree that the permission is granted to the application. For example any GPS information would be guarded by the *Location* permission. An application accessing any system resources has to use the appropriate API functions to interact with the Android kernel. A subset of API functions is associated with the respective Android permission.

Semantic Associations of UI Elements and API Calls. Recent research concentrated on the relations between UI elements to the API calls which are triggered by interacting with this element. Backstage [2] is a static analysis tool, which is able to mine associated APIs over a big set of applications and determine outliers based on this mapping.

The approach is designed to link single elements to executed API calls, i.e. Backstage statically observes for any UI element which API calls are triggered. In a second step, it groups UI elements from a large scope of apps by their semantic word similarity to identify which elements use atypical API calls. As an example, 'sign up' and 'register' would be grouped into the same semantic cluster. This allows to re-identify the semantic meaning of an UI element in different applications, even if the exact button label varies. However, it is not designed to investigate dynamic behavior, i.e. it does not consider the context in which an action is executed.

There is ongoing research in the area of semantic web application analysis to capture the general topics behind the individual steps of the user interactions for automatic cross-app test generation [7]. This research leverages natural language processing techniques to identify a similarity value between textual elements. The same semantic similarity measurements shall be utilized for my proposed research.

Application State Model. In the most simple case, the behavior of an application can be modeled by a sequence of UI screens (referred to as states), connected by the user interactions (modeled as transitions leading from one state to another).

In previous research this basic model was enhanced to allow for hidden behavior (not observable by the user) to be encoded in this model. For instance permission event graphs (PEGs) [4] have demonstrated that there are repeating patterns on when and in which order certain API calls or event handlers are normally triggered.

Automated Application Exploration. Automatic testing and exploration techniques [6] have become pretty popular. They interact with the user interface of an application to explore all reachable screens and thus dynamically produce a state model of the application under test. In other words, either a given test case can be used to navigate through the different features of an application or random based interactions can be executed. These interactions together with the visited screens form an *execution trace* through the application state model.

3 METHODS

The proposed research is based on the concept of user expectations. A user expectation entails the desire to access a specific feature and the expectation of the triggered behavior. The accessed feature can be concluded from the UI context, i.e. the button label and the screen context. Meanwhile the user expects that the actual app behavior conforms with the implied functionality.

This means, *additional semantic meta information* is necessary to construct a sufficiently expressive behavioral model. In particular, the dynamically created model has to be enhanced with semantic concepts for each state, i.e. for each presented screen. To derive these concepts, I plan to investigate various possibilities.

As a starting point, the presented content can be encoded as *semantic word vectors*. Meaning, the semantic context of each present UI element is one entry in the vector and two states represent the same semantic concept if the entries of their vectors are semantic similar.

Another possibility is to extract a topic for the present contents of each state and utilize the similarity of these topics to determine if two states belong to the same concept.

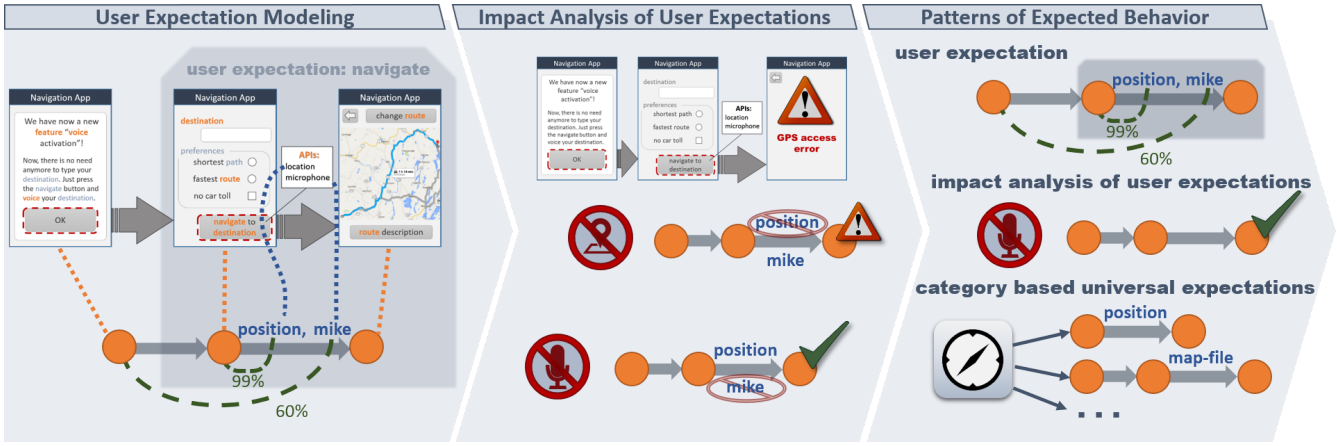

Figure 2: Process Pipeline to identify behavioral patterns related to user expectations. User expectations are modeled as a sequence of screens, each containing semantic information for the UI elements and their respective API calls. For these APIs an impact analysis is performed by blocking the API calls and analyzing the effect on the model.

3.1 Modeling User Expectations

Following the assumption that users have certain expectations when using an application is not far fetched. The previous 'navigate' example in Figure 1 puts the user expectation into context. The user acknowledges or is intuitively aware that using a navigation framework requires access to the current location. From a privacy preserving point of view, it is also legit to assume that the application only needs access to the location, when it is actively used. Technically, this means that the access to the location is allowed, when the UI element 'navigate to destination' is clicked, until the destination guidance is stopped. In other words, the expectation when using this function is linked to a specific observable behavior.

The shortest of such user expectations involves exactly one action, e.g. clicking a certain button. Likewise, user expectations can span a sequence of interactions within the application model. Therefore, the idea is to define user expectation based on such execution trace. In particular, the observed semantic concepts have to be considered together with the intermediate actions, i.e. the triggered APIs and used arguments.

The research challenge tackled in this scope is to determine *to which extent a particular action is expected in the context of the related UI element text, the current screen content and a sequence of semantic states and actions.*
To address this question, my idea is to associate the triggered API call with a semantic concept, reflecting the purpose of this function. The API calls guarded by permissions are a good starting point to reflect security sensitive actions and the semantic concepts could be concluded from the permission categories. But in the long-term this set is extended towards an extensive set, i.e. by including recent research results for security sensitive APIs with a set of fine-grained semantic concepts.

In our navigation example (Figure 2) the API call accessing the GPS would be associated with the concept 'position' and the microphone APIs with 'mike'. A semantic analysis is used to determine the similarity between the UI text 'navigate to destination' and 'position'. This specific comparison would result in a high similarity value. In contrast, the similarity to 'mike' would be very low, meaning it is very unlikely to be expected. In that case, it is necessary to consider the remaining contents of the current state and the previous states, to determine if any UI content indicated the action. That way, it is possible to detect that the previous screen presented the user with information of a 'voice activation' feature. The microphone usage can be concluded from the semantic context of the previous state.

To derive a value of how far certain behavior can be expected, I plan to weight the different similarity results based on the distance from the respective UI content to the actually triggered action. Furthermore, it is desirable to find a threshold value for the similarity to determine when the behavior is part of the user expectation and when the correlation is simply too weak, such that a user would not expect the triggered behavior.

3.2 Impact Analysis of User Expectations

Intuitively it is clear that any executed security relevant API call presents a potential risk as sensitive data or hardware components are accessed. Still, the user desires to access certain features of an application which require unexpected API calls. As a non-expert, the user will likely accept the access. Thus, an impact analysis on what feature is not available, if certain API calls are not allowed, allows to decide *if the feature warrants the use of sensitive APIs.*

For this purpose, the impact on the user observable behavior is analyzed when security sensitive API calls are blocked. If no feature is affected, this intuitively indicates non-essential behavior, i.e. over-privileges or even hidden (malicious) behavior. Other than that, it is possible to identify which states in the behavioral model and therefore which consecutive features, related to user expectations, are affected by the considered transaction. By mining these effects over multiple applications within the same category, patterns for behavioral features shall be identified.

A camera app, for example, typically has a button 'record' which triggers an access to the camera to produce a video or picture and present the result on the screen. If the camera access is blocked,

the application will likely end up in a different (maybe in an error handling) state. Thus the pattern of a *record and display* feature can be associated to the respective sensitive action in the model.

3.3 Patterns of Expected Behavior

The app store has a multitude of applications offering similar features, typically organized in categories. The implementation behind these apps might differ, but if a user downloads an arbitrary navigation app, he expects that this app offers a certain set of core functions, e.g. the possibility to navigate from A to B. As already mentioned, it can also be assumed that the downloaded app is expected to use the location manager, if a route is to be generated.

For testing a newly downloaded app, I plan to use the models of apps within the same category as a ground truth for the expected behavior. I.e., universal user expectations can be extracted from other apps and these patterns should be present in the new app as well. In other words, *specifications* can be learned by analyzing a multitude of applications.

These specifications express *how an application is supposed to behave from the users perspective* and hence can be used as rules for enforcement systems. The current state-of-the-art only allows to manually specify custom policies, which are created for every app. This task is non-trivial for an average user. In contrast, the novel approach allows to automatically infer how any application is expected to behave and this expectation can be expressed as a policy.

A detailed look into an execution trace, reveals which semantically equivalent UI elements have been interacted with and what behavior was entailed by these actions. For the app under test, the observed traces can be compared with the universal expectations to identify outliers for the triggered behavior or the user observable impacts of security sensitive actions. This gives an idea of the default behavior for this application category and corresponds to the intuition that user expectations are complemented by the experience with other applications.

4 EVALUATION

There is no state of the art research trying to identify user expectations based on the presented UI content and to the best of my knowledge no tools exist to automatically generate user expectation based policies either. Hence, there is no reference to be directly compared with the presented approach. However, there is ongoing research in the area of malware classifiers, which aim to classify the applications API behavior and to assess privilege usage. The new approach of expectation based behavior analysis is general enough that it should be able to cover this purpose.

The question, if user expectations reflect the applications behavior (RQ1), can be verified by checking if they are usable to differentiate between benign and malicious behavior.

In order to evaluate how suitable user expectations are to express normal and benign behavior (RQ4), the approach is applied on a set of malicious applications. The derived expectation based policies are then evaluated against the results of state of the art classifiers. The assumption is that most of the malicious behavior can be considered to be unexpected and therefore should be covered by the generated policies.

Furthermore, to evaluate the quality of category based universal expectations (RQ2), it is reasonable that different functionalities should result in distinctive user expectations. Therefore, the universal expectation based patterns are investigated for category specific expectations. The intuition is that particular functionalities should be recognizable by certain behavioral patterns in the model. If the Play Store categories are too coarse grained to reflect the variety of functionalities, the applications could be categorized based on their meta information instead, e.g. the application description.

Finally, I want to evaluate if user expectations can serve as an oracle (RQ3). The user expectations are linked to available features altogether with the necessary privileges and semantic context. Meaning, if some of the required privileges are blocked or essential UI elements are mutated, certain expectations should no longer be satisfied. The resulting oracle employs reachability of user expectations to decide if a respective test succeeds to fulfill a specific expectation.

5 EXPECTED CONTRIBUTIONS

In this paper, I present a novel approach to analyze application behavior from the users perspective. In particular user expectations are inferred from execution traces and the observed UI content. These expectations are mined over multiple applications to obtain universal user expectations which are applicable across applications. With these patterns at hand, it is possible to identify surprising behavior on the UI level. It can be determined if the actual application behavior is reasoned by the presented UI context or is regarded as hidden behavior. Moreover, user expectations can be used for testing, to determine if an execution trace fulfills a specific expectation or to measure how many user expectations are covered by a set of executions.

REFERENCES

[1] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining Apps for Abnormal Usage of Sensitive Data. In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*.

[2] Vitalii Avdiienko, Konstantin Kuznetsov, Isabelle Rommelfanger, Andreas Rau, Alessandra Gorla, and Andreas Zeller. 2017. *Detecting Behavior Anomalies in Graphical User Interfaces.* Technical Report. Saarland University and IMDEA Software Institute of Madrid. https://www.st.cs.uni-saarland.de/appmining/backstage/backstage_tech_report.pdf

[3] Michael Backes, Sven Bugiel, Sebastian Gerling, and Philipp von Styp-Rekowsky. 2014. *Android security framework : enabling generic and extensible access control on Android.* Technical Report. Saarländische Universitäts- und Landesbibliothek. http://scidok.sulb.uni-saarland.de/volltexte/2014/5734

[4] Kevin Zhijie Chen, Noah Johnson, Vijay D'Silva, Shuaifu Dai, Kyle MacNamara, Tom Magrino, Edward Wu, Martin Rinard, and Dawn Song. 2013. Contextual Policy Enforcement in Android Applications with Permission Event Graphs. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS '13)*.

[5] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *ICSE'14: Proceedings of the 36th International Conference on Software Engineering*.

[6] Konrad Jamrozik, Philipp von Styp-Rekowsky, and Andreas Zeller. 2016. Mining sandboxes. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 37–48.

[7] Andreas Rau. Topic-Driven Testing *(ICSE'17 PhD Symposium)*. Saarbrücken, Germany.

[8] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1105–1116.

[9] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *In IEEE Symposium on Security and Privacy*. 95–109.