PvI 27-11 (1) Best lang stuk over waarom app sec. een issue is, maar komt niet echt to-the-point...

1. Introduction

onderstatement? Maar ook cliche

Draft version: 3, ext. reviews: FS

Mobile applications play an ever-more important role in our lives. They are the gateways to use social media, to perform banking transactions, to submit working hours and much more. The same? applications are not only limited to run on mobile phones, but can also run on other smart devices such as smart televisions, -watches and -cars. This is in particular true for applications that run on Android, since it is the most popular operating system for smart devices[1]. Although the platform is well established, the security of its applications is not. This attribution especially became true when last year researchers of the Norwegian firm Promon were able to exploit Tesla's Android application and achieve full control of the Tesla car paired with the application[1]. A few months after, the research institute Fraunhofer SIT found exploitable vulnerabilities in 9 popular password managers for Android that could compromise the passwords currently stored by the user [2]. These episodes show that although today's society moves towards a pervasive adoption of mobile applications, the applications are insufficiently secured.

jargon

oorzaak-gevolg?

which? draait Tesla ook op Android? Relatie?

locally?

examples

These examples illustrate

only? Main?

The reason why applications deficit security is because software security in general always has been part of a tradeoff where development methodology and time-to-market play an essential role. An early time-to-market brings an economical strategic advantage for companies because of two reasons. First of all, when a company launches an application as soon as possible for a certain niche domain, they have the ability to become market leaders, with the ability of locking in users. Secondly, publishing software expeditiously also generates an early revenue. On the other hand a security mature development methodology, such as test driven development, might consume more time thus delaying the time-to-market, but results in a more secure application.

literatuur ref? Is dit wel relevant?

(1) ↑↑

Modern security testing tools aid the process of discovering bugs by applying a multitude of automated testing techniques that comes in three flavors: black-box, white-box and grey-box testing. White-box testing tests the application's internal logic by code reviews or specific tests. Grey-box testing tests the software's logic using metadata, such as documentation or file structure. Black-box testing treats the software as an interactive application and determines whether the correct output is given for the correct input.

3 flavors → referentie? examines

ik weet niet of ik mij in definitie hen vinden. Heb je hier ref. voor?

[1] https://developer.android.com/about/android.html

Volgens mij:
- black: outside-in, zonder voorkennis (i.e. alleen binaries)
- grey: outside-in, met bijv. credentials
- white: inside-out, met volledige kennis.

(2) Zorg voor goede intro over State machine learning. What is het? Wat kan het?

(3) Brug is er niet doordat intro ontbreekt. Volgens mij kan structuur worden: belang app sec > vulns in Android apps > technieken voor automated vuln scanning > state machine learning t.b.v. vuln scanning.

(2) wat?

Black-box testing techniques can also be fine-tuned to understand the application's internal logic. This is what a state machine learning algorithm does, by observing a large number of traces: a combination of inputs and outputs. The inferred state machine reveals a lot of information about the application's logic, and might as such function as a data source for identifying bugs or vulnerabilities in the application. This has been achieved for a wide range of software systems, such as various TLS driver implementations [3]. The inferred models were assessed for extraneous transitions or states and paths that could bypass security measures were found. Another way to use state machine learning is to develop a specification for an implementation when there exists none. This has been performed for the Dutch biometric passport, where it was more time efficient to infer a state machine than having a team of experts establish such a model [4]. @Rick & Paul, de vorige 2 paragrafen vormen de brug van testing (black box testing) naar active learning. Ik heb het gevoel dat deze brug nog iets groter gemaakt kan worden. Misschien iets vertellen over hoe KPMG mobile testing doet? Of hebben jullie andere suggesties?

komt uit de lucht vallen. Introduceren.

zins opbouw.

Beetje random voorbeelden, doordat intro ontbreekt.

(3)

Model learning can be done on a set of existing traces (*passive learning*) or generate the set of traces while learning by interacting with an application (*active learning*). The drawback of passive learning is that the model is complete in the variety of existing traces, i.e. when certain application behavior is not described in the set of traces, the inferred model does not describe this behavior either. Active learning overcomes this problem by querying for the information it needs to know. Since software systems, and as a subset to that group Android applications, are able to respond interactively, they can function as a data source to generate these traces. The drawback of active learning in this situation is that interacting with an application consumes time, as each input and output combination needs to be simulated. In order to improve the learning process, different active learning algorithms have been developed.

zins opbouw klopt niet. If incomplete??

loopt niet lekker

which situation?

(4)

eerst even opsomming doen?

Active state machine learning is often implemented according to the MAT framework, which composes the entities of a learner and a teacher. The goal of the learner is to infer the state machine model of a system under test (SUT), by asking membership queries and equivalence queries to the teacher. Membership queries ask whether a certain behavior input is recognized by the SUT. The query and answer combination together form a trace, and a hypothesized model can be established after a sufficient amount of traces are generated. The learner then queries an equivalence query to the teacher for the established hypothesis, which determines if the model correctly describes the SUT. If this is the case, learning halts as a correct model has been inferred. If the model incorrectly describes the SUT's behavior, a trace will be provided by the teacher which distinguishes the model and the SUT. The process repeats itself until an equivalence query yields success. This process is visually depicted in Figure 1.1.

→ afko?

"correct"? sufficiently correct?

(4) Lees dit alles nog eens kritisch door!

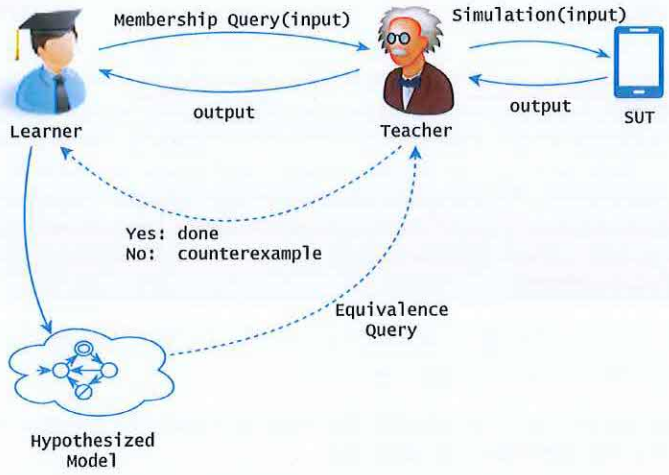

Figure 1.1: Active Learning with a Learner and a Teacher

A key component of the discussed framework is the ability for the teacher to verify the equivalence between the hypothesized model and the model implemented by the SUT. Establishing an oracle that is able to verify equivalence is impossible [5], because this model itself is nonexistent. Equivalence between an hypothesized model and the SUT is often established through approximation, where the teacher generates a number of test cases and verifies whether the model's output is equal to the SUT. If a given test case yields a different result, the model is inequivalent and the test case itself forms a counterexample. There are numerous algorithms that generate these test cases like randomized input sequences and the W-method [6]. A common drawback of the test case generation algorithms is that the tests are insufficiently diverse or excessively long. Test case generation is a study on its own and this thesis will touch upon various algorithms to be applied for equivalence approximation.

Because the model that is inferred by active state machine learning manifests additional information about the application, the model might also be used as a new data source to assess the application's security. Up until now, most research involving active learning stops when the model is automatically inferred and continues with manual model inspection for the search of extraneous behavior. The security assessment that is concluded from the manual inspection thus also depends on the reviewer and may yield different results for different reviewers. Furthermore research on retrieving such a model for mobile Android applications is very limited as there is only one study performed by Lampe et al. that developed a tool for a specific application to infer a state machine model [7].

The aforementioned research provides a limited framework for model inference of mobile applications. Due to its limitations, the framework can function as a basis for this research to first of all overcome these limitations and therefore utilize the inferred model as a data source for an automated security assessment.

1.1 Outline

This thesis conducts research on how to infer a correct state machine model for a generic mobile Android application and assess its security in an automated way. To infer a correct model one has to review active automata learning algorithms and solve challenges such as the equivalence approximation between a model and an application. Furthermore, algorithms that identify vulnerabilities on the input of an inferred model need to be established. The primary goal of this thesis is to utilize these building blocks to answer the following main research question of this thesis:

How can one identify weaknesses in mobile Android applications through feasible behavioral state machine learning?

To aid the process of answering the main question, the question has been divided into the following sub-questions:

RQ 1. How can one extend model learning to be applicable to mobile Android applications?
This research question mainly focusses on reducing or mitigating the limitations of the framework provided by Lampe et al. This question deals with the issues that arise when introducing active learning to the mobile application domain, such as the definition of a complete alphabet and identification of mobile parameters.

RQ 2. How can we improve the feasibility of model learning of Android applications? As it has already been mentioned, active learning from simulations is time consuming. Different active learning algorithms reduce the time complexity by limiting the number of queries and the overall query length. This question focuses on the different learning algorithms and corresponding attributes such as model equivalence approximation.

RQ 3. How can the learned model be used to assess the application's security? The novelty of this thesis lies in the application of active learning on Android applications and the automatic processing of the model as a new data source to assess the application's security. The latter inquires a set of identification algorithms that determine the presence of a certain vulnerability on input of the inferred model.

The initial expectation would be that model inference of Android applications can be achieved by extending the framework proposed by Lampe et al. One should note that the tool has not been maintained since publication, so starting the tool with all the dependencies is already a starting requirement. Moreover, the hypothesis would be that the inferred model can function as a data source for identifying vulnerabilities, but this requires additional contextual information, such as the type of API calls. Given the assumption that the inferred model describes the entire application, certain invariants for the contextual information items must hold, such as all API calls must be on SSL.

The structure of this thesis is as follows. Chapter 2 gives an overview of the building blocks of active model inference, where various active learning algorithms are discussed as well as techniques to assure model conformance. Chapter 3 reviews the prior work from Lampe et al. by elaborating on the framework they proposed and identifying its limitations. Chapter 4 establishes requirements to overcome the identified limitations and proposes a solution framework that is developed based on these requirements. Chapter 5 establishes algorithms that identify security vulnerabilities in the learned models. Chapter 6 depicts the results of the residual proof of concept running on various mobile Android applications. Chapter 7 discusses these results, answers the research questions and provides a perspective ahead on future work references. This thesis closes by the conclusion stated in Chapter 8.

Bibliography

[1] Lars Lunde Birkeland. Tesla cars can be stolen by hacking the app. `https://promon.co/blog/tesla-cars-can-be-stolen-by-hacking-the-app/`, 2016.

[2] Steven Arzt Stephan Huber, Siegfried Rasthofer. Extracting all your secrets: Vulnerabilities in android password managers, 2017.

[3] Joeri De Ruiter and Erik Poll. Protocol State Fuzzing of TLS Implementations. In *USENIX Security Symposium*, pages 193–206, 2015.

[4] Fides Aarts, Julien Schmaltz, and Frits Vaandrager. Inference and abstraction of the biometric passport. *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 673–686, 2010.

[5] Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. Learnlib: a framework for extrapolating behavioral models. *International Journal on Software Tools for Technology Transfer (STTT)*, 11(5):393–407, 2009.

[6] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, May 1978.

[7] KQ Lampe, JCM Kraaijeveld, and TD Den Braber. Mobile application security: An assessment of bunq's financial app. *Delft University of Technology Research Repository*, 2015.