Driving Apps to Test Third-Party Component Security

Jaeyeon Jung
Microsoft Research
jjung@microsoft.com

Collaborators



Ravi Bhoraskar, Seungyeop Han, & David Wetherall
University of Washington



Shuo Chen, Suman Nath, & Rui Wang
Microsoft Research



Jinseong Jeon
University of Maryland, College Park



Tanzirul Azim
University of California, Riverside

3rd-party components used commonly



38% of apps

41% of installs

http://www.appbrain.com/stats/libraries/ad *last updated*: Jul 02, 2014



12% of apps

25% of installs

http://www.appbrain.com/stats/libraries/social *last updated*: Jul 02, 2014

3rd-party components commonly used

admob

millennialmedia
The Mobile Advertising Platform

FLURRY



Scoreloop



http://www.prlog.org/10983527-hipaa-security-risk-analysis.jpg



9
Like
45
Tweet
1
+1
1
Share
0
Pin it
submit

SECURITY

Update vulnerability in third-party SDK exposes some Android apps to attacks


Lucian Constantin

Dec 10, 2013 1:50 PM

A third-party advertising framework integrated in hundreds of Android apps contains a vulnerability that could allow hackers to steal sensitive information from users' phones, according to security researchers from antivirus firm Bitdefender.

The framework is called HomeBase SDK (software development kit) and is developed by Widdit, based in Ramat Gan, Israel. It allows Android developers to monetize their apps by displaying ads and custom content on the phone's lock screen.

HomeBase SDK is one of the many mobile advertising frameworks available to Android developers, but according to the Bitdefender researchers it has an insecure update mechanism that could put users at risk when their phones connect to the Internet over insecure or compromised networks.

Every time an application that uses HomeBase is executed, the SDK code embedded into it connects to a remote server using an unencrypted connection to get the latest version of the SDK, the Bitdefender researchers said Tuesday in a blog post. If an updated SDK is found it is downloaded in the form of a JAR (Java Archive) file and executed, they said.

+1 0 Tweet 0 Like 0
CONNECT WITH US

Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization

Authors:
Rui Wang, *Microsoft Research Redmond;* Yuchen Zhou, *University of Virginia;* Shuo Chen and Shaz Qadeer, *Microsoft Research Redmond;* David Evans, *University of Virginia;* Yuri Gurevich, *Microsoft Research Redmond*

Open Access Content

Papers are restricted to registered attendees until the event begins. Once the event begins, the content becomes free and open to everyone. Journal articles are open to everyone upon publication. If available, video, audio, and/or slides of this presentation will be posted here after the event.

Wang PDF View the slides BibTeX

Abstract:
Most modern applications are empowered by online services, so application developers frequently implement authentication and authorization. Major online providers, such as Facebook and Microsoft, provide SDKs for incorporating authentication services. This paper considers whether those SDKs enable typical developers to build secure apps. Our work focuses on systematically explicating implicit assumptions that are necessary for secure use of an SDK. Understanding these assumptions depends critically on not just the SDK itself, but on the underlying runtime systems. We present a systematic process for identifying critical implicit assumptions by building semantic models that capture both the logic of the SDK and the essential aspects of underlying systems. These semantic models provide the explicit basis for reasoning about the security of an SDK. We use a formal analysis tool, along with the semantic models, to reason about all applications that can be built using the SDK. In particular, we formally check whether the SDK, along with the explicitly captured assumptions, is sufficient to imply the desired security properties. We applied our approach to three widely used authentication/authorization SDKs. Our approach led to the discovery of several implicit assumptions in each SDK, including issues deemed serious enough to receive Facebook bug bounties and change the OAuth 2.0 specification. We verified that many apps constructed with these SDKs (indeed, the majority of apps in our study) are vulnerable to serious exploits because of these implicit assumptions, and we built a prototype testing tool that can detect several of the vulnerability patterns we identified.

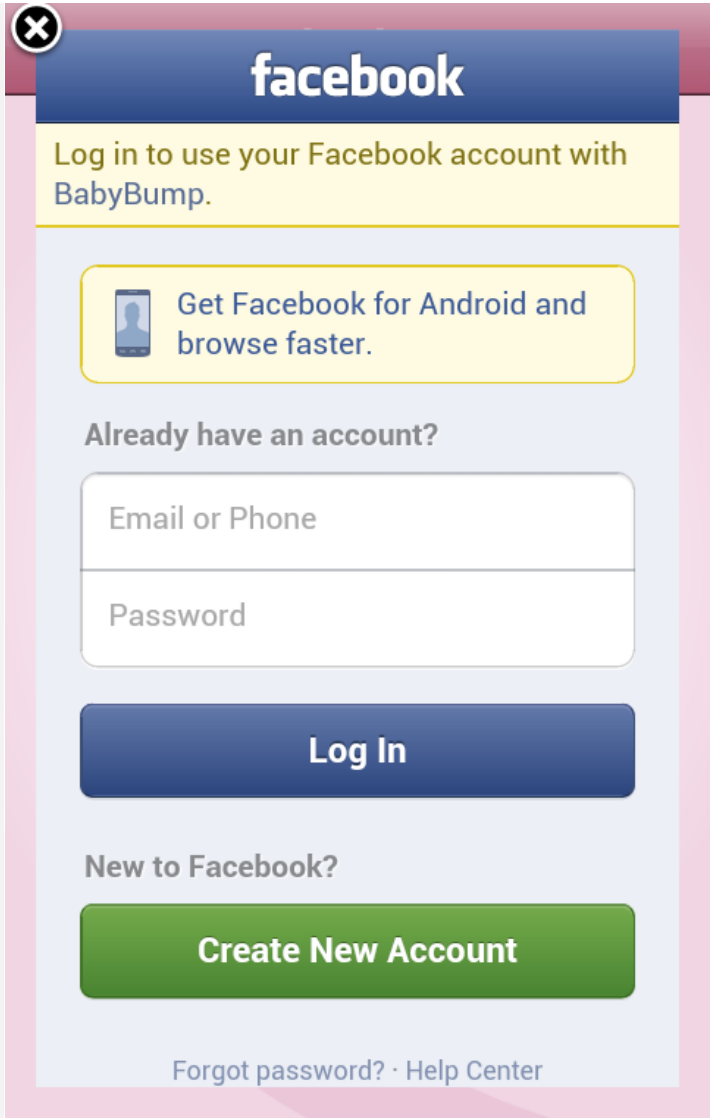

brahmastra

Goal

Test 3rd-party component integration at scale

Support app store operators, security researchers to test a large number of apps using the same third-party component for a potential vulnerability

Develop tools that enable testers to observe in situ interactions between the third-party component and remote services in the context of a specific app at runtime





Key findings

Brahmastra enabled us to successfully analyze third-party code in 2.7x more apps than the state-of-the-art GUI testing tool:

(1) A study of 220 apps with ad component shows that 80% kids apps displayed ads that attempt to collect personal info and 36% show inappropriate content

(2) A study of 200 apps with Facebook Connect shows that 13 popular apps were still vulnerable to the access token attack



Problem & Insights



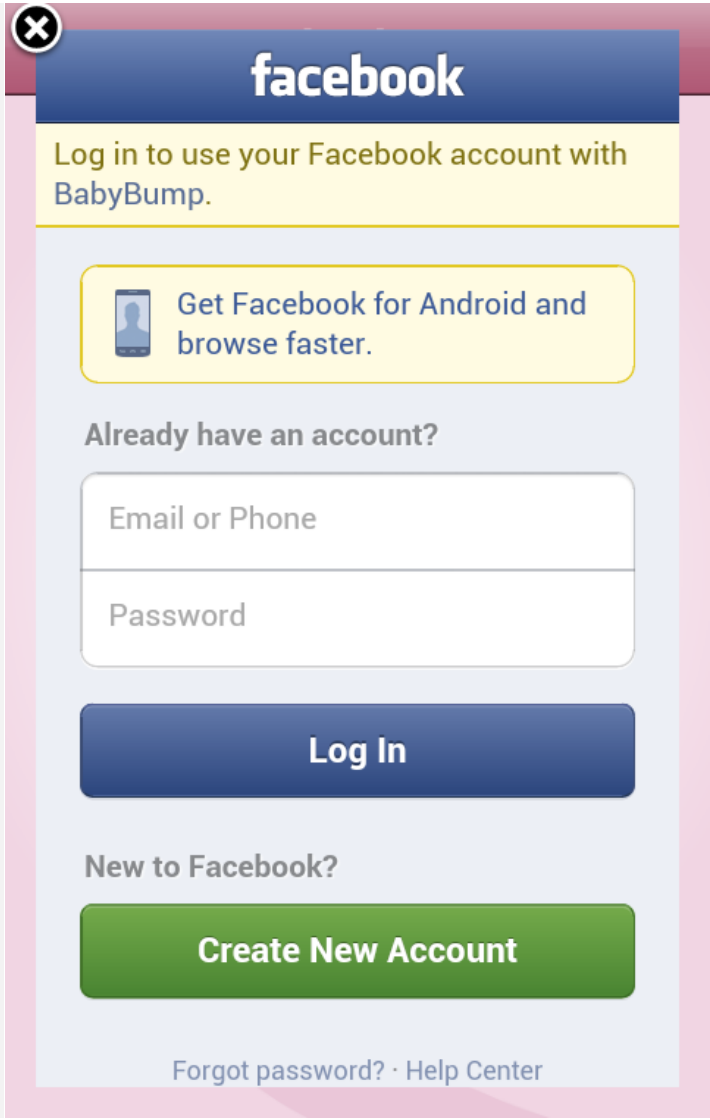
Example

Is this app vulnerable to Facebook access token attack [Wang13]?

(1) Navigate through the app to open Facebook login page

(2) Provide login credentials and click “Log In”

(3) Examine network messages between the app, the app server, and the Facebook server





Why not using Monkeys?

Poor hit rates

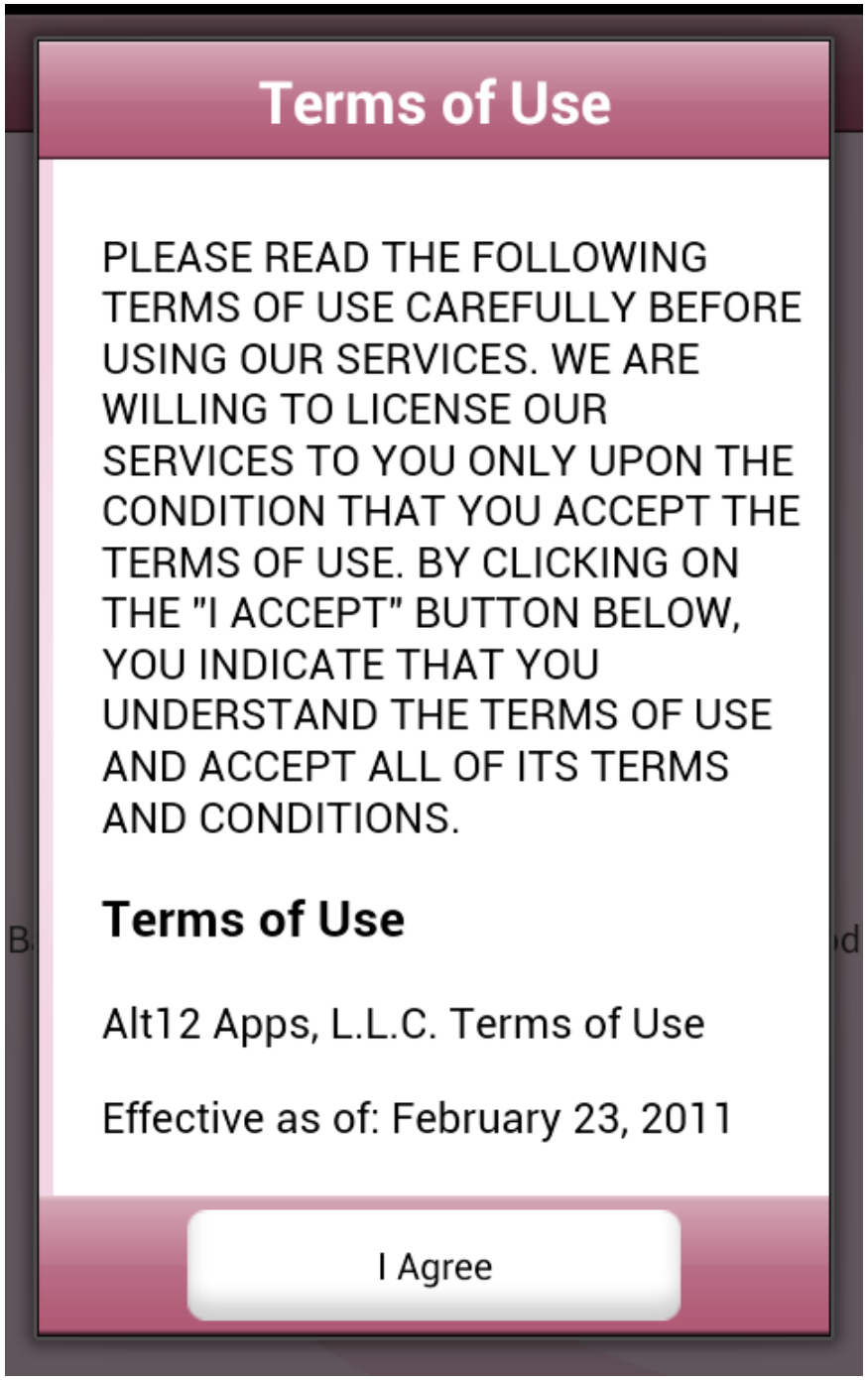
(1) Low code coverage

(2) Unable to provide semantically correct inputs

(3) Failing to explore custom controls and UI elements that are not visible in the current screen

(4) Crashing apps by overly stressing the UI







A1:.`settings.NameDueDate2`

E1: click “I Agree”





A1:.`settings.NameDueDate2`

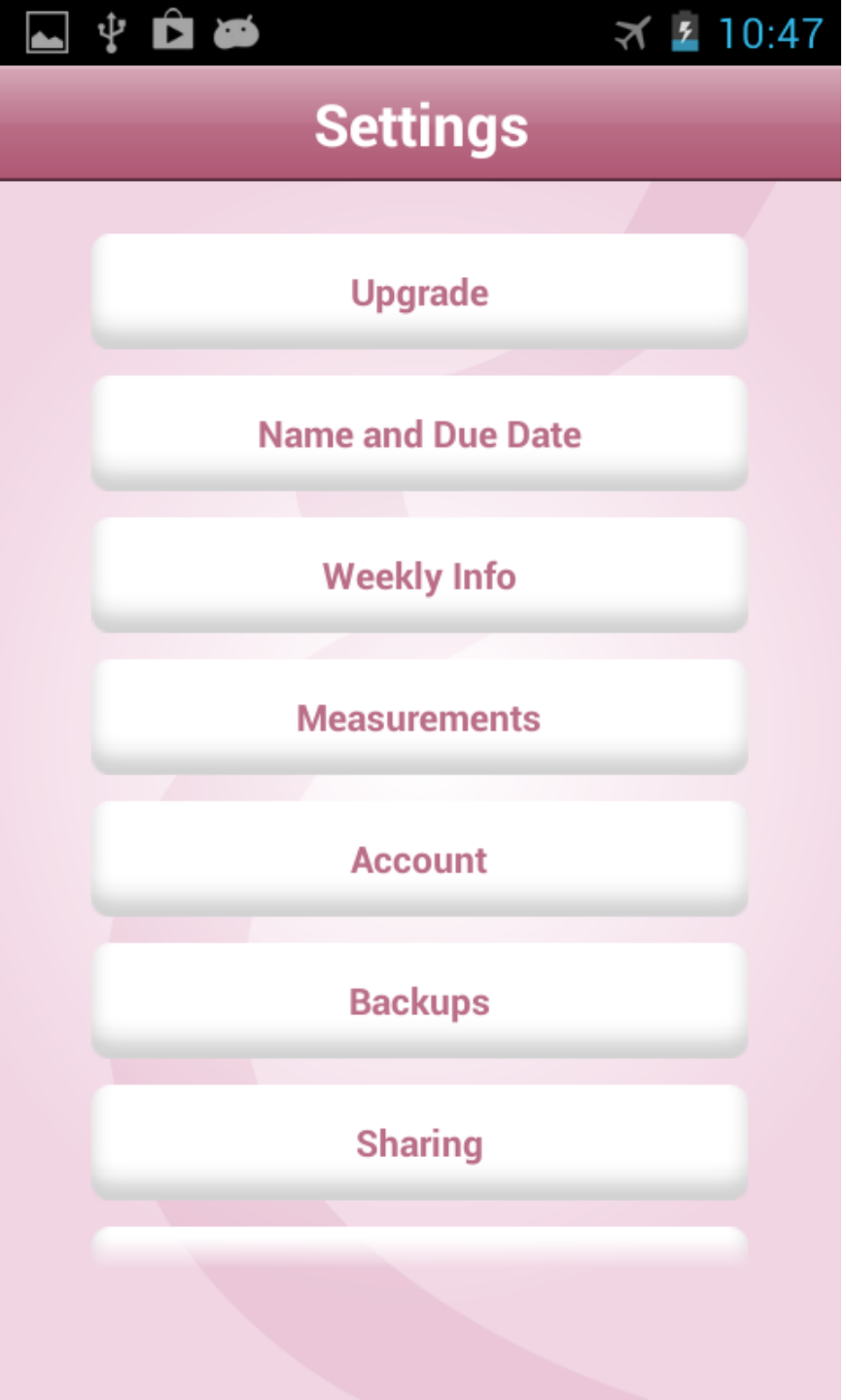
E2: click “Done”





A2:`.babybumpcore.Main`

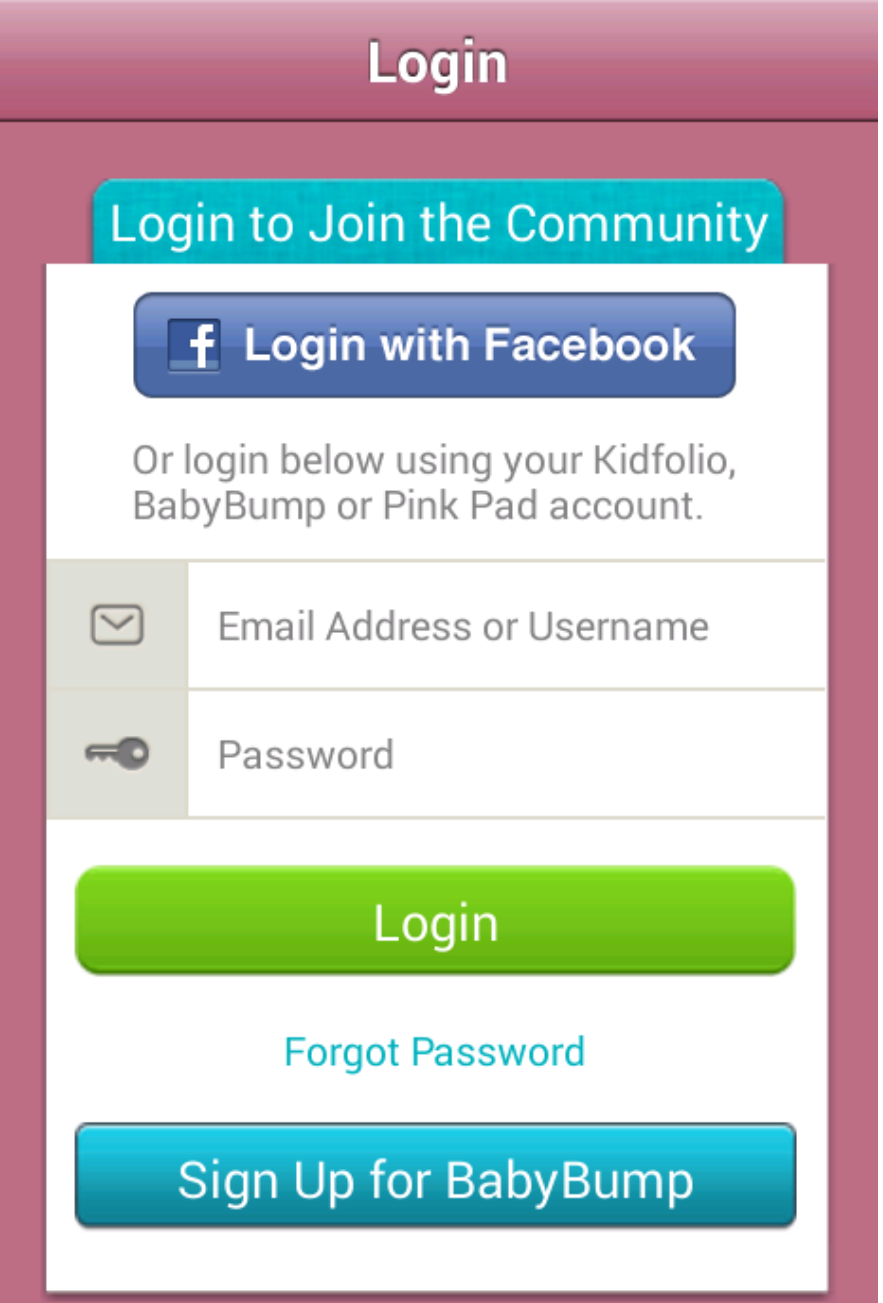
E3: click “Settings”

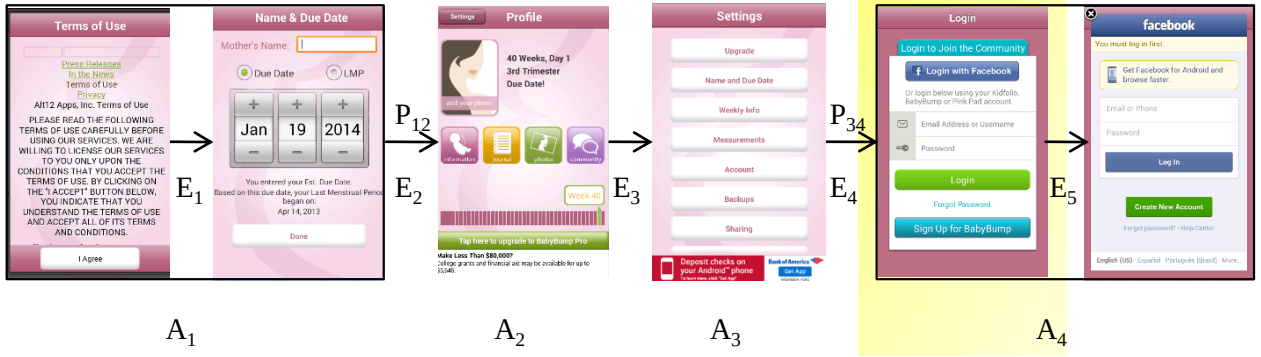




A3:`..settings.Settings`

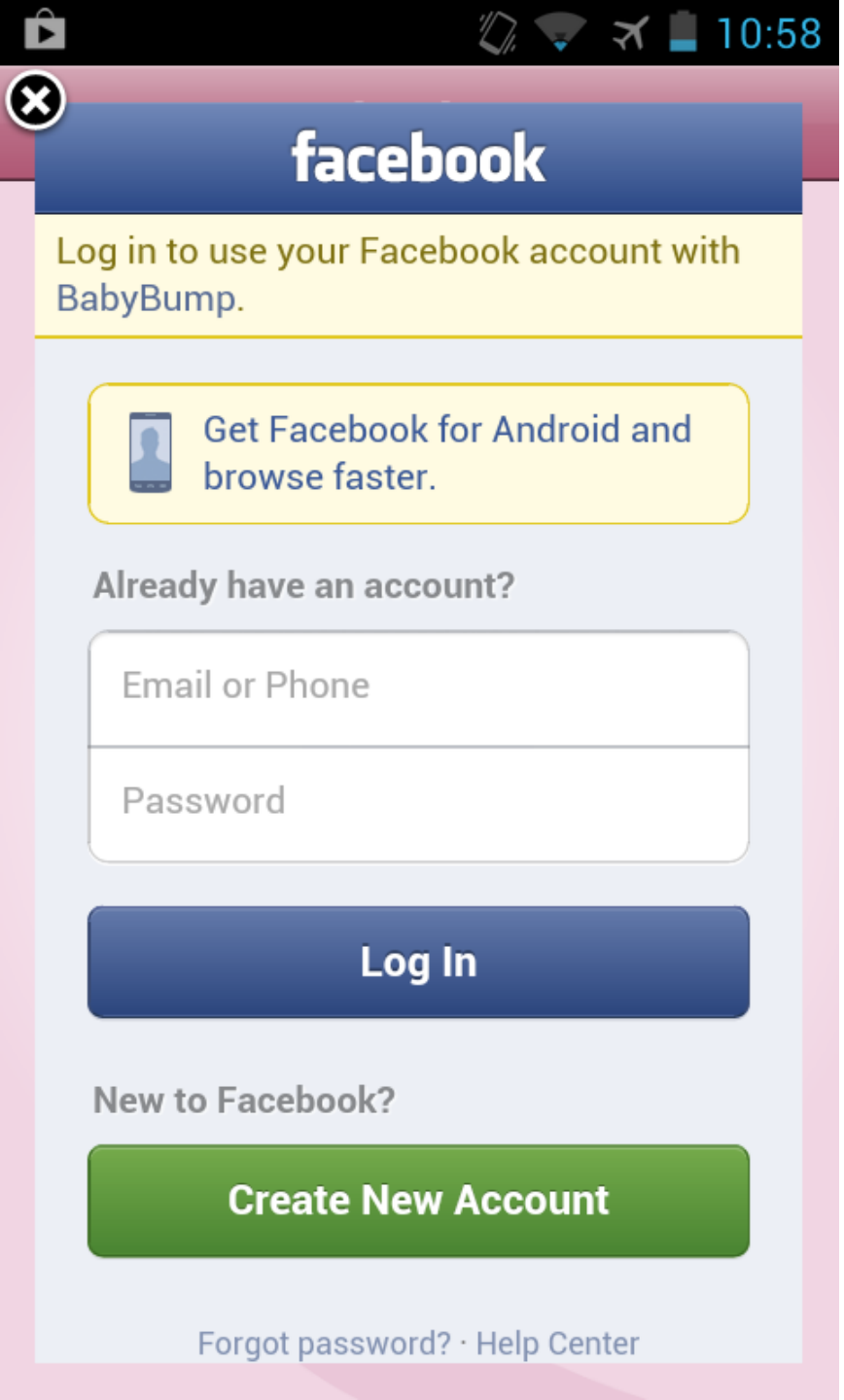
E4: click “Account”

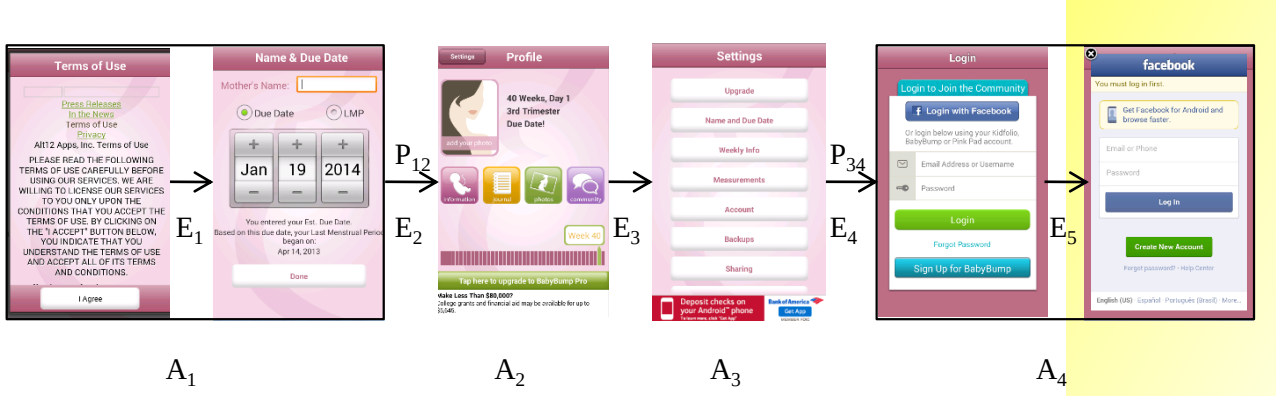




A4:`.activity.LoginActivity`

E4: click “Login with Facebook”





A4: `.activity.LoginActivity`

E4: type in credentials and click “Log In”

Our approach

Leverages the structure of Android apps to improve test hit rate and execution speed

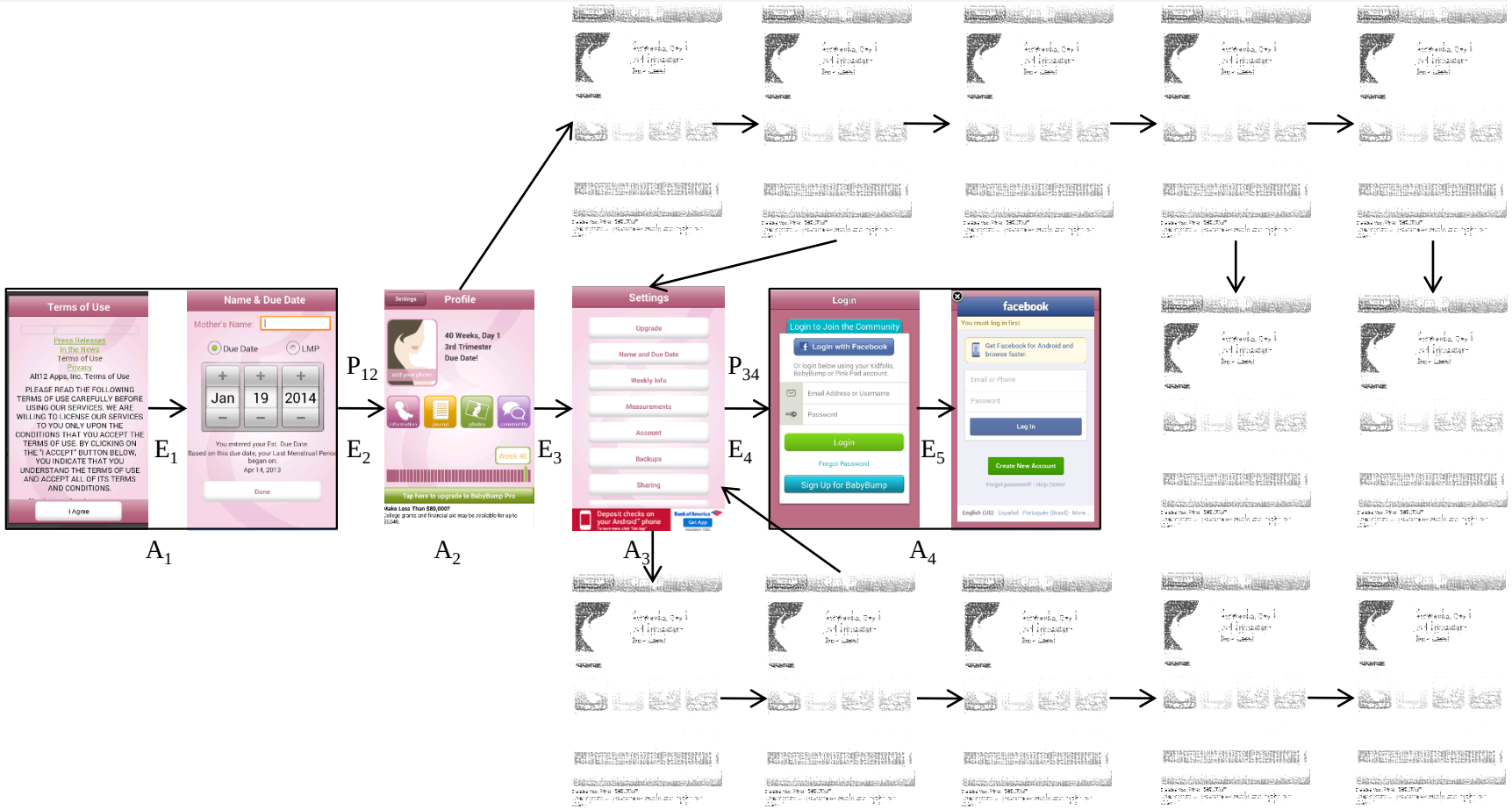
Characterize an app by statically building a page transition graph and call chains

Rewrite the app under test to directly invoke the callback functions that trigger the desired page transitions



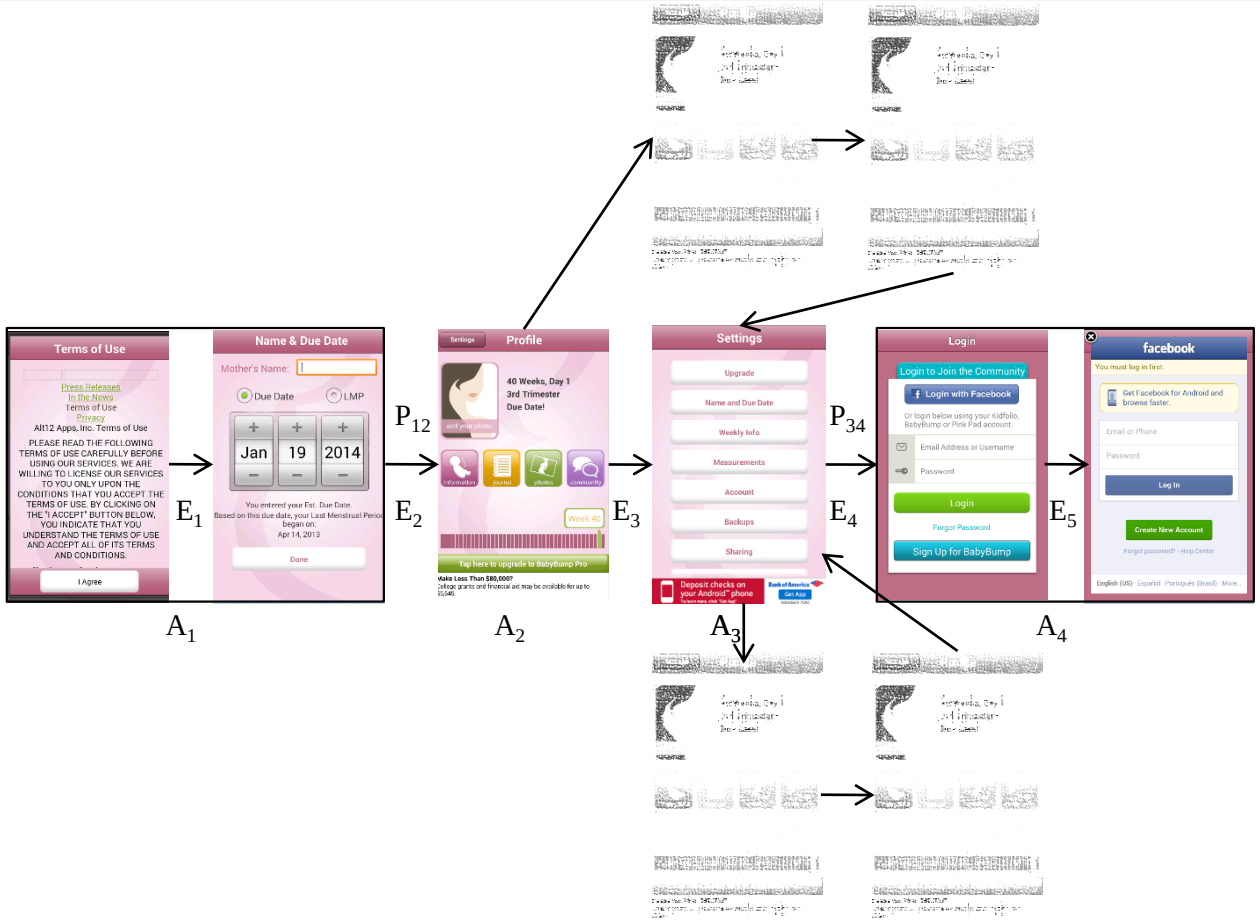
Brahmastra vs. Monkeys

(1) Static path pruning



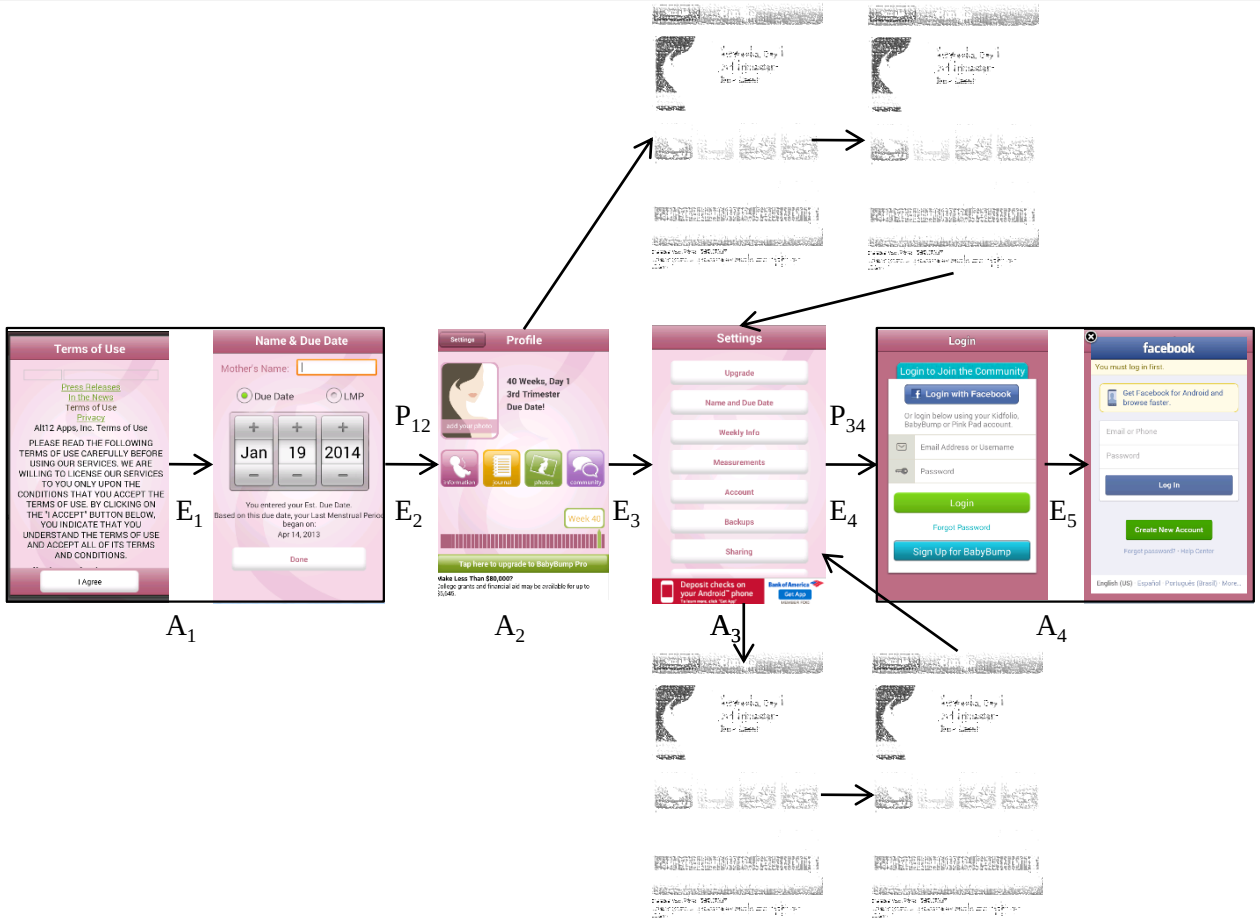
Brahmastra vs. Monkeys

(1) Static path pruning: A1->A2->A3->A4



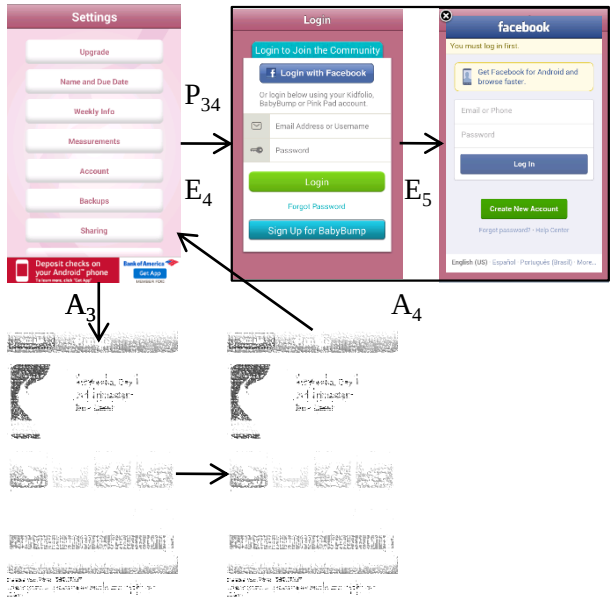
Brahmastra vs. Monkeys

(2) Dynamic node pruning



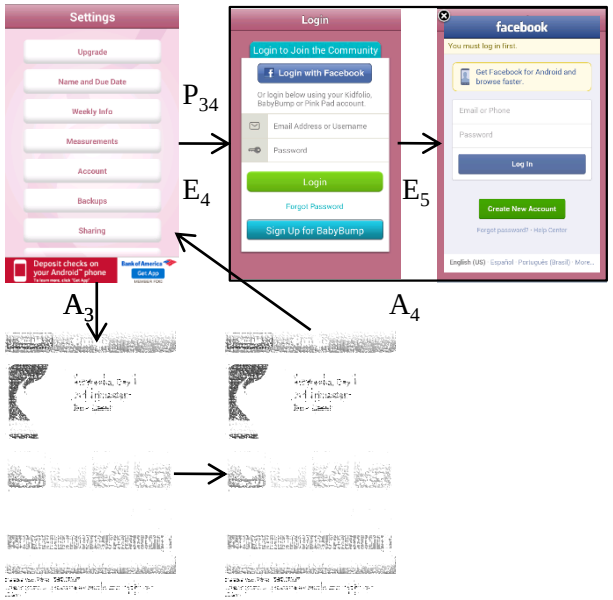
Brahmastra vs. Monkeys

(2) Dynamic node pruning: A3->A4



Brahmastra vs. Monkeys

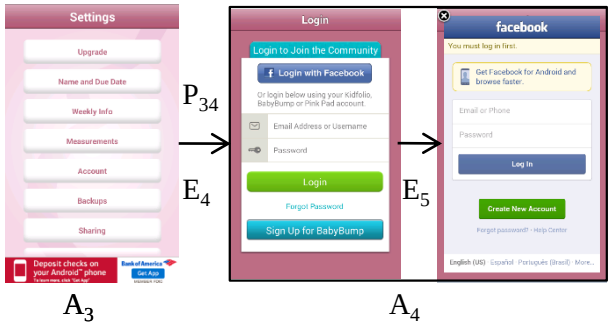
(3) Directed self-execution





Brahmastra vs. Monkeys

(3) Directed self-execution: A3 (E4) -> A4





Design & Implementation



Execution Planner

Discover an execution path to invoke the target 3rd-party method

(1) Determining target activity(s)

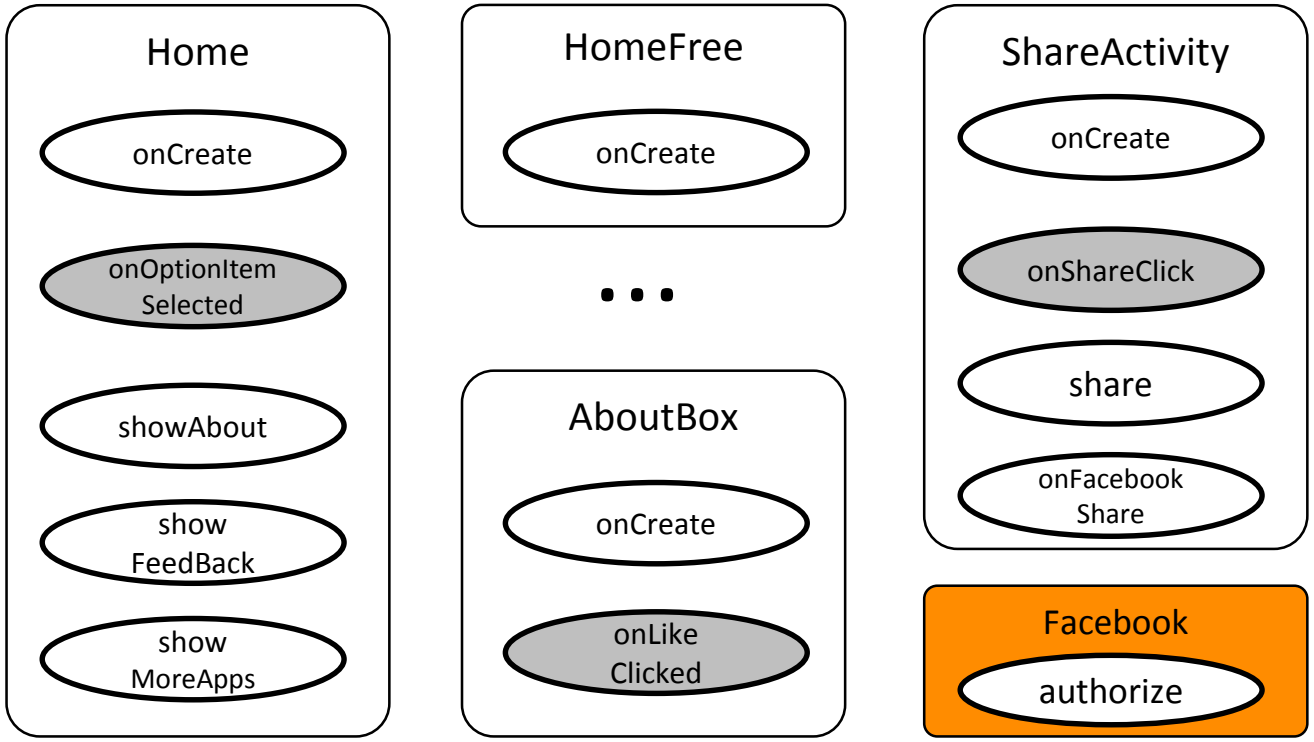
(2) Constructing partial call graphs first

(3) Finding activity transition paths

(4) Efficient call graph computation by searching the transition graph backwards

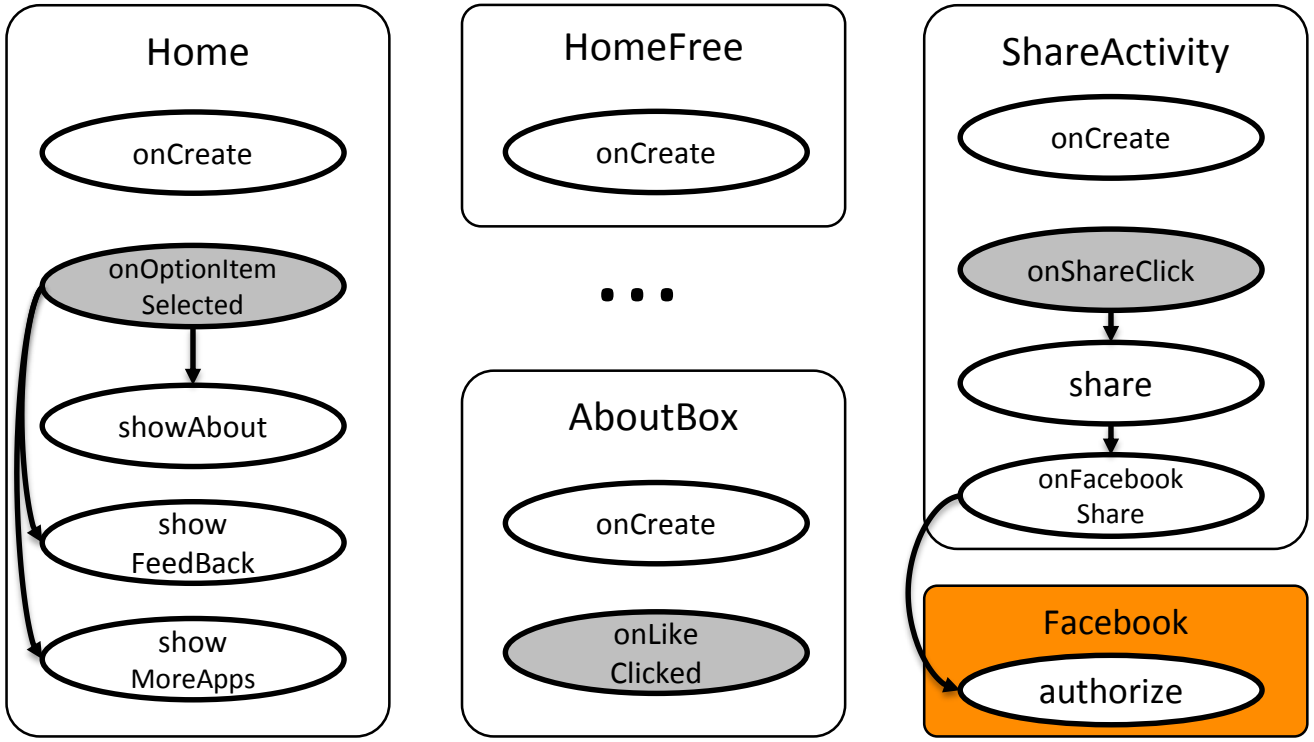


Execution Planner: call graph





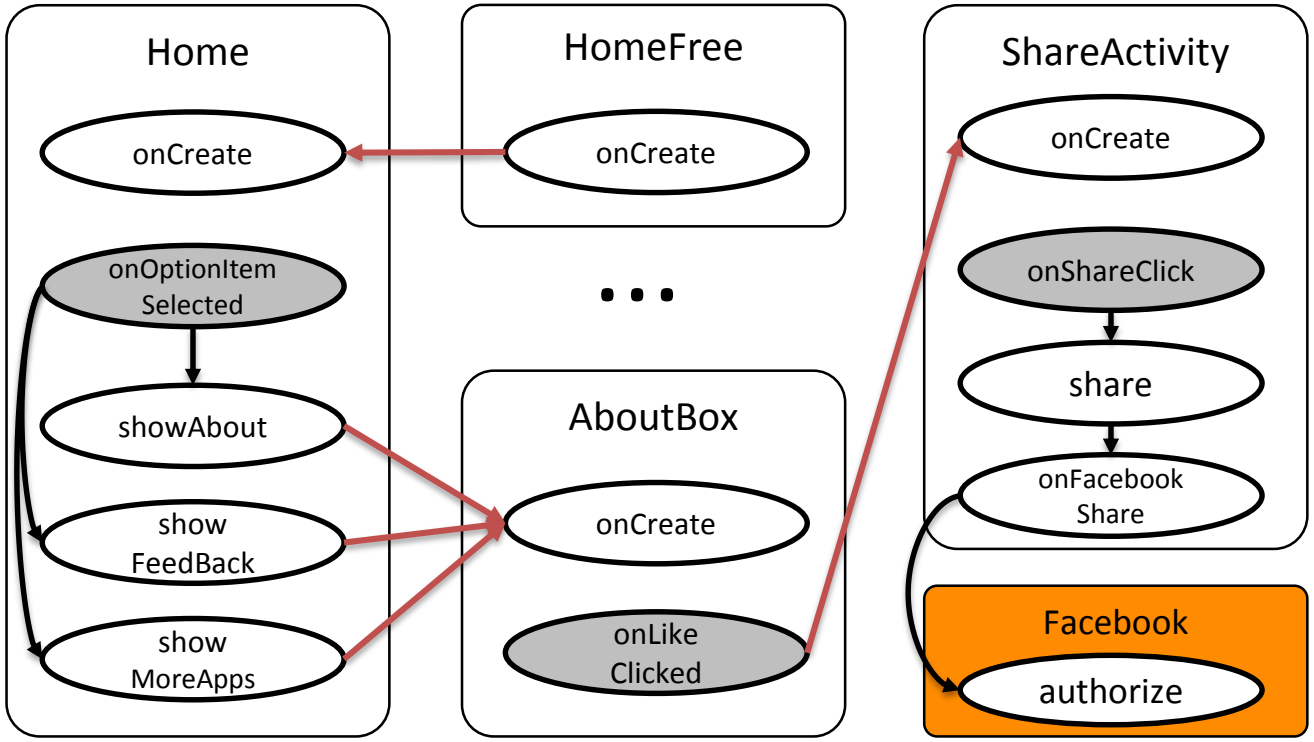
Execution Planner: call graph



(1) Synchronous calls



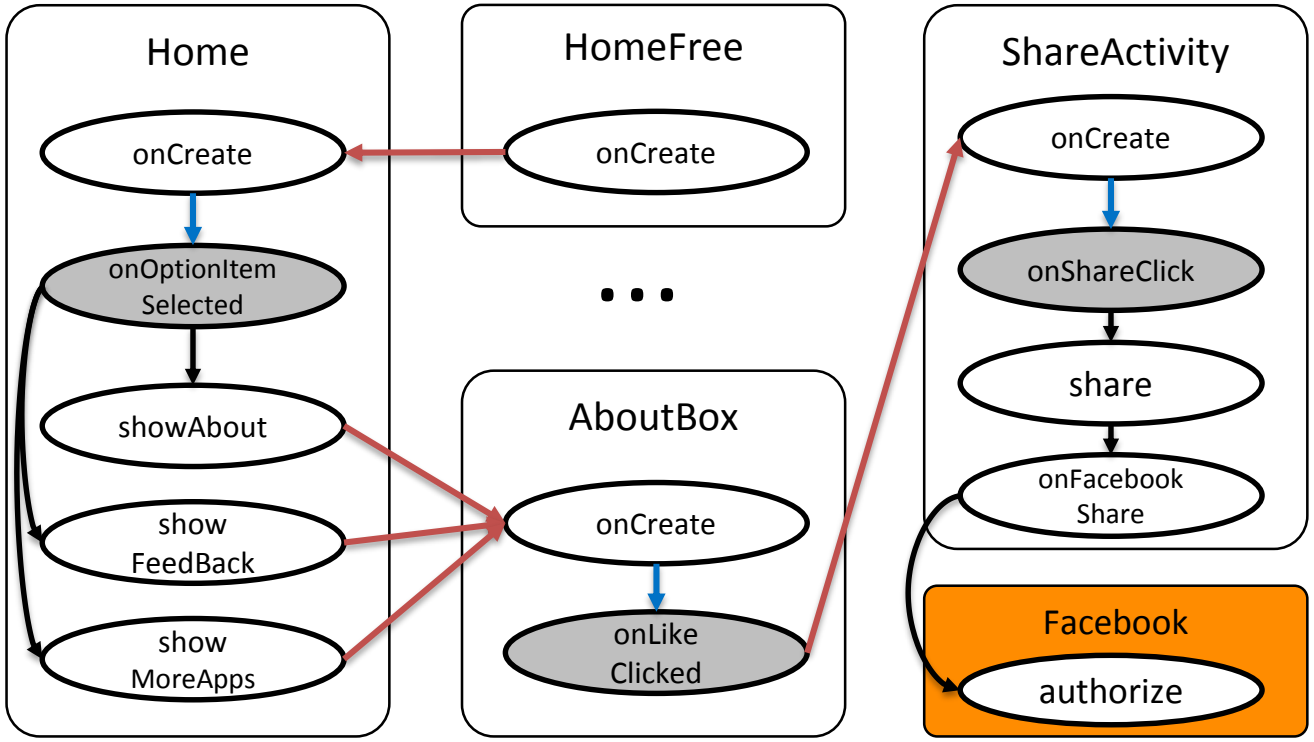
Execution Planner: call graph



(1) Synchronous calls

(2) Activity transitions

Execution Planner: call graph

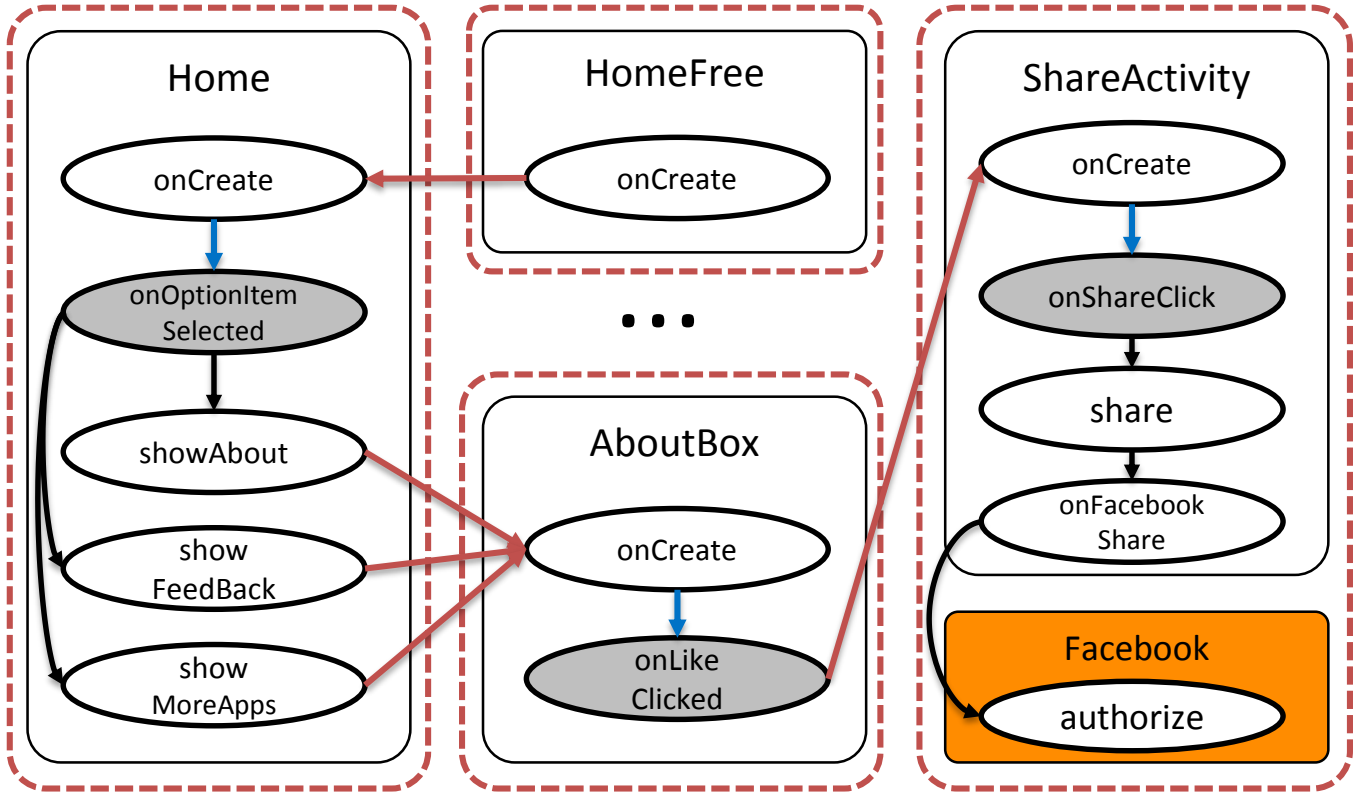


(1) Synchronous calls

(2) Activity transitions

(3) Implicit calls due to user interactions

Execution Planner: call graph



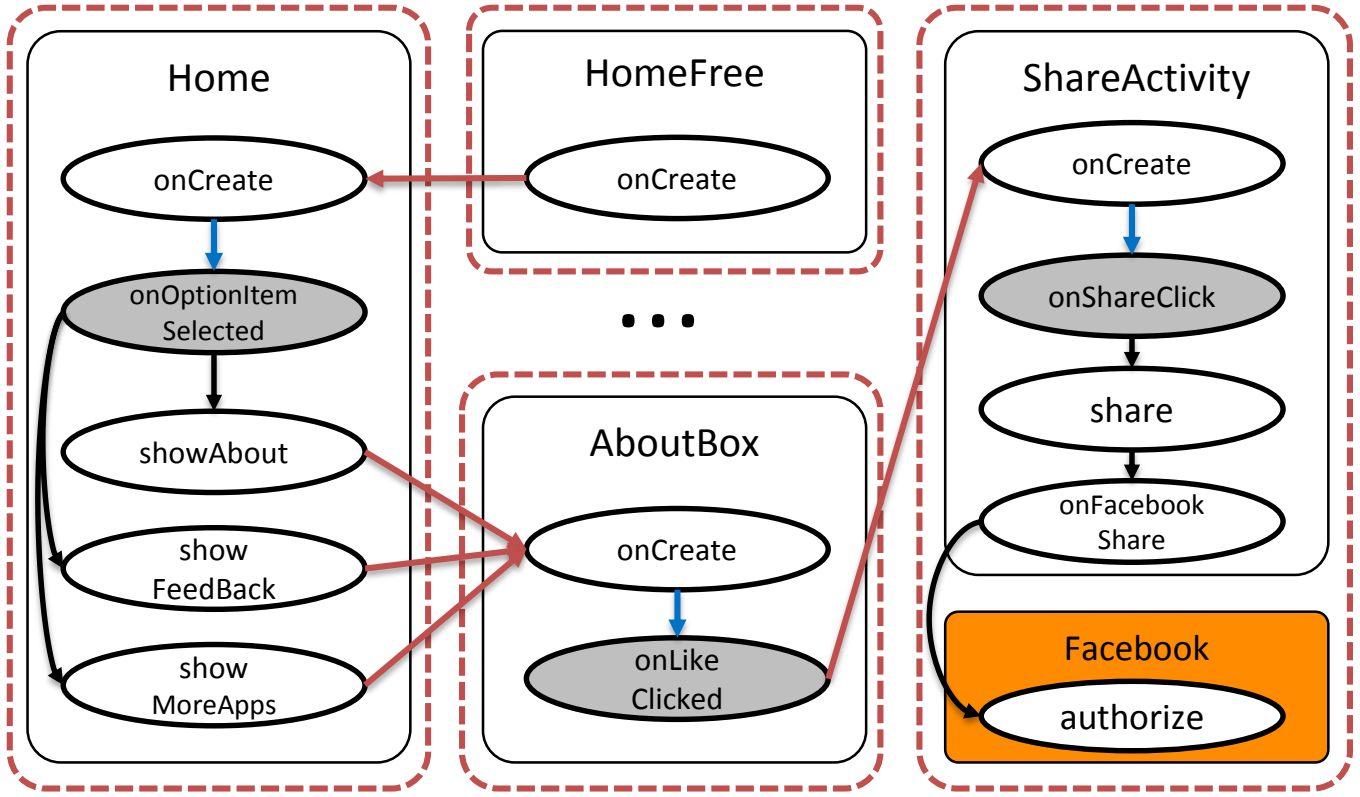
(1) Synchronous calls

(2) Activity transitions

(3) Implicit calls due to user interactions



Execution Planner




```
   HomeFree; .onCreate
-> Home; .onCreate
-> Home; .onOptionsItemSelected
-> Home;. showAbout
-> AboutBox; .onCreate
-> AboutBox; .onLikeClicked
-> ShareActivity; .onCreate
-> ShareActivity; .onShareClick
-> ShareActivity; .share
-> ShareActivity; .onFacebookShare
```

https://github.com/plum-umd/redexer



Execution Engine

(1) Rewriting an app using Soot in order to artificially invoke a callback method upon the completion of the exercising the current activity

(2) Using ADB to perform jump start using Activity Manager after rewriting the app's Manifest file

Evaluation of Brahmastra



Methodology

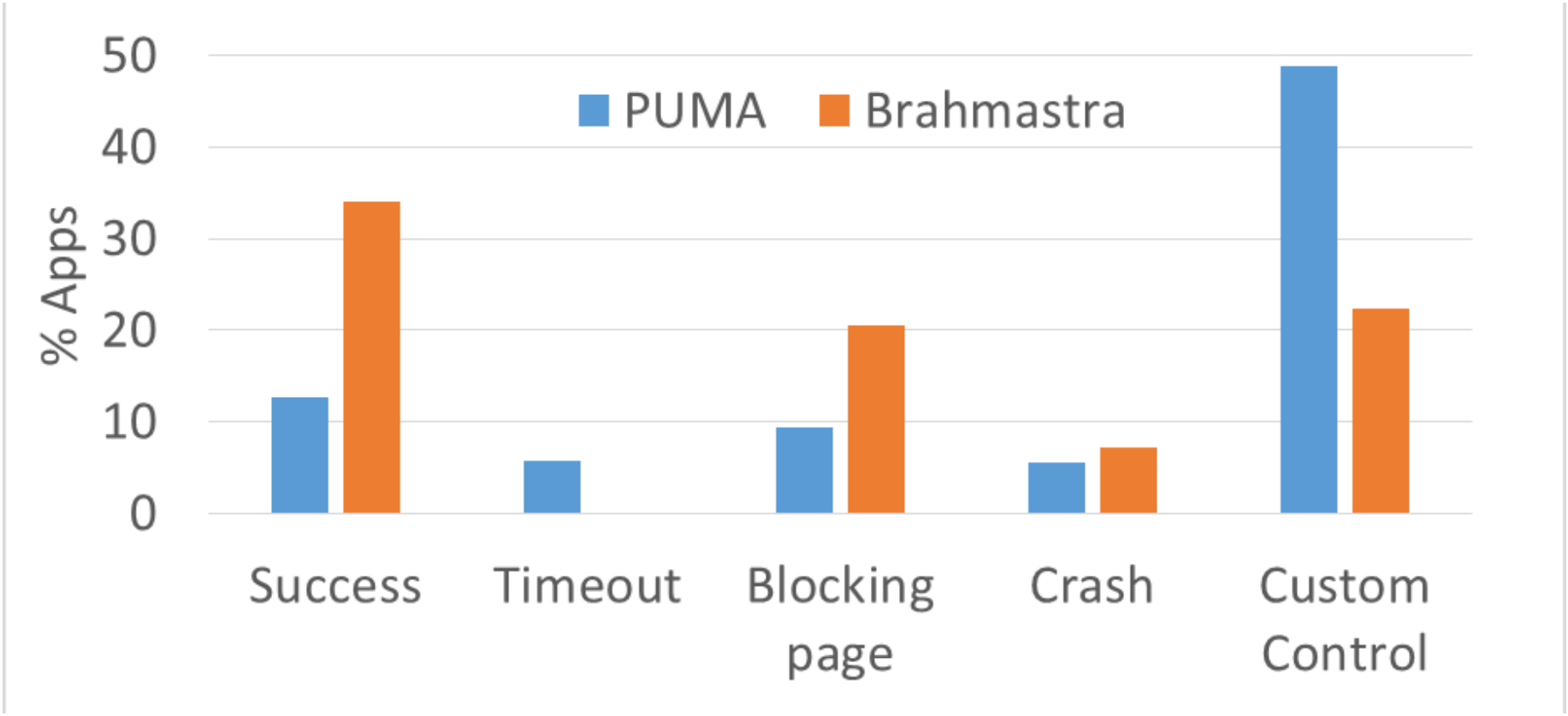
(1) Target method: authentication methods in the Facebook SDK for Android

(2) Apps: 1,010 popular apps crawled from Play Store

(3) App execution: 5 short paths (Brahmastra); 250 steps of random exploration (PUMA)

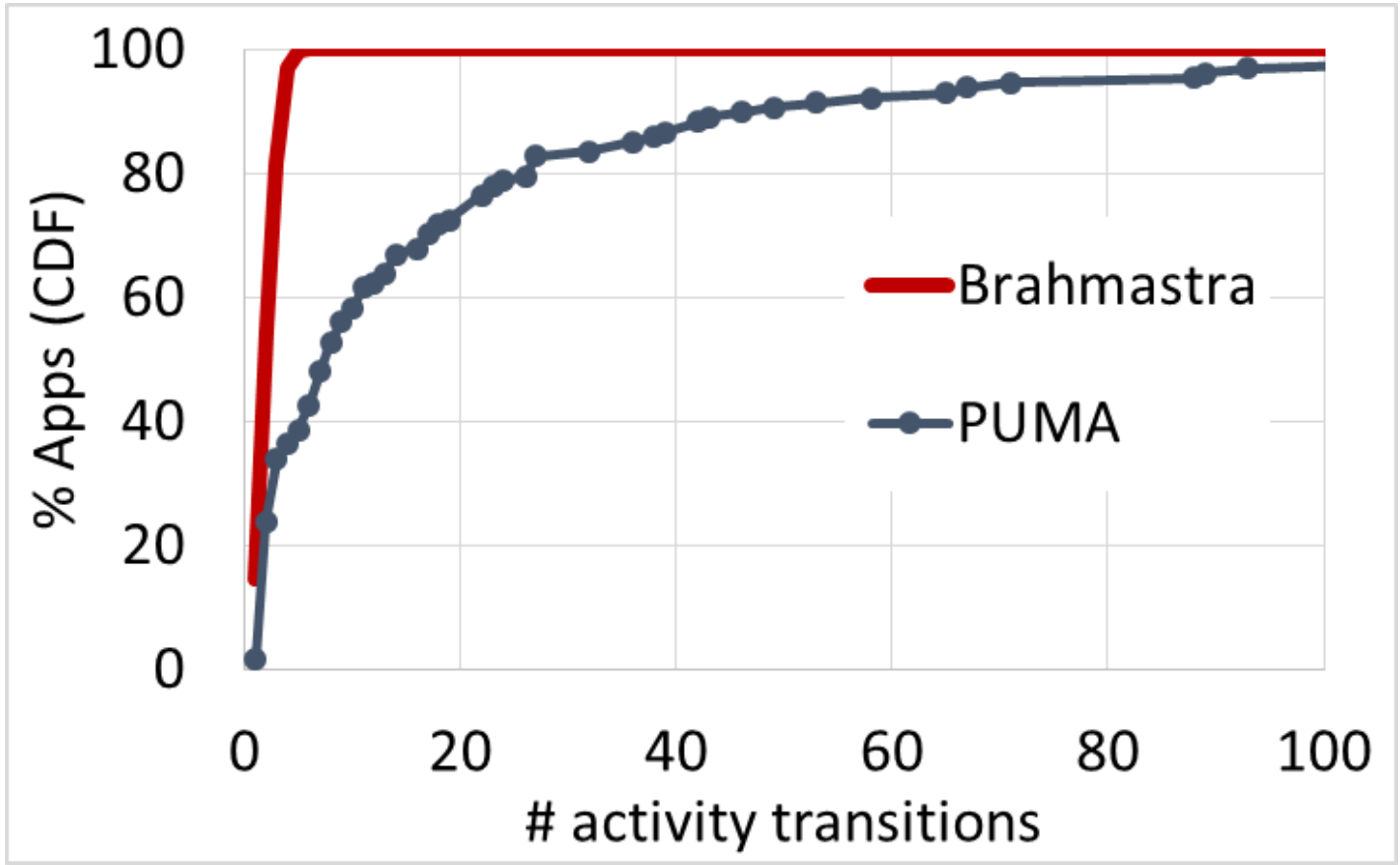
Hit rate

127 apps (13%) with PUMA vs. 334 apps (34%) with Brahmastra



Test speed

18.7 (PUMA) vs. 2.5 (Brahmastra)





Security Analysis #1: Ads in Kids Apps

COPPA

The Children's Online Privacy Protection Act

(1) mobile app developers must follow the stipulations if their apps are directed at children under 13 years old

(2) disallows the collection of personal information by these app unless the apps have first obtained parental consent.

Test goal

To measure the extent of potentially non-COPPA compliant ad components in kids apps:

(1) whether in-app ads or landing pages pointed by these ads present forms that collect personal information?

(2) whether content displayed in in-app ads or landing pages is appropriate for children?



Test procedure

(1) Driving apps to display ads

Used Brahmastra to invoke API that loads ads of 220 apps categorized by "Kids" in Amazon Android app store

Focused on apps with AdMob or Millennial Media

(2) Collecting ads & landing pages

Used a Fiddler proxy to collect URLs

(3) Analyzing ads & landing pages

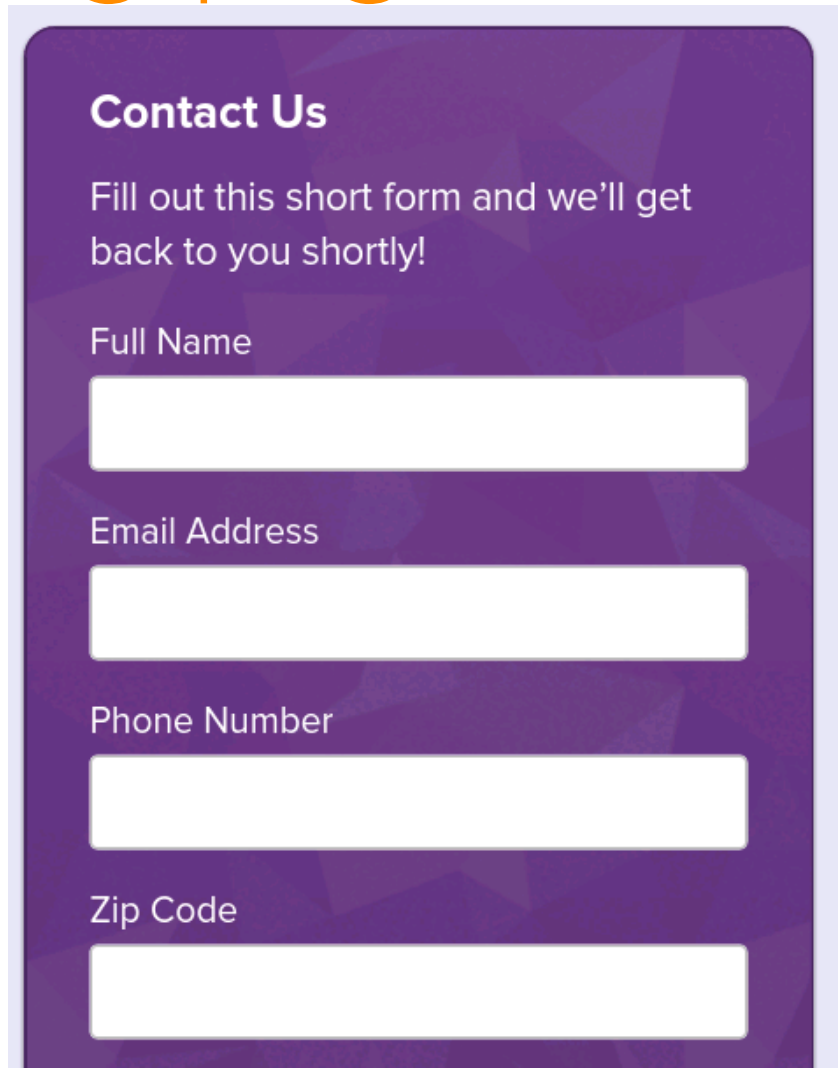
Probed the Web of Trust (WoT) DB to get the "child safety" score of each URL

Used Amazon Mechanical Turk to characterize each page



Results: types of personal info. collected by landing pages

Information Type	% apps
Home Address	26%
First and last name	79%
Online contact	42%
Phone number	7%
TOTAL	80%



Results: child-inappropriate content in ads

Content Type	% apps
Child Exploitation	3%
Gambling	1%
Misrepresentation	7%
Violence, weapons or gore	2%
Alcohol, tobacco, drugs	2%
Profanity and vulgarity	0%
Free Prize	26%
Sexual Content	13%
TOTAL	36%





Security Analysis #2: Social Media Add-ons

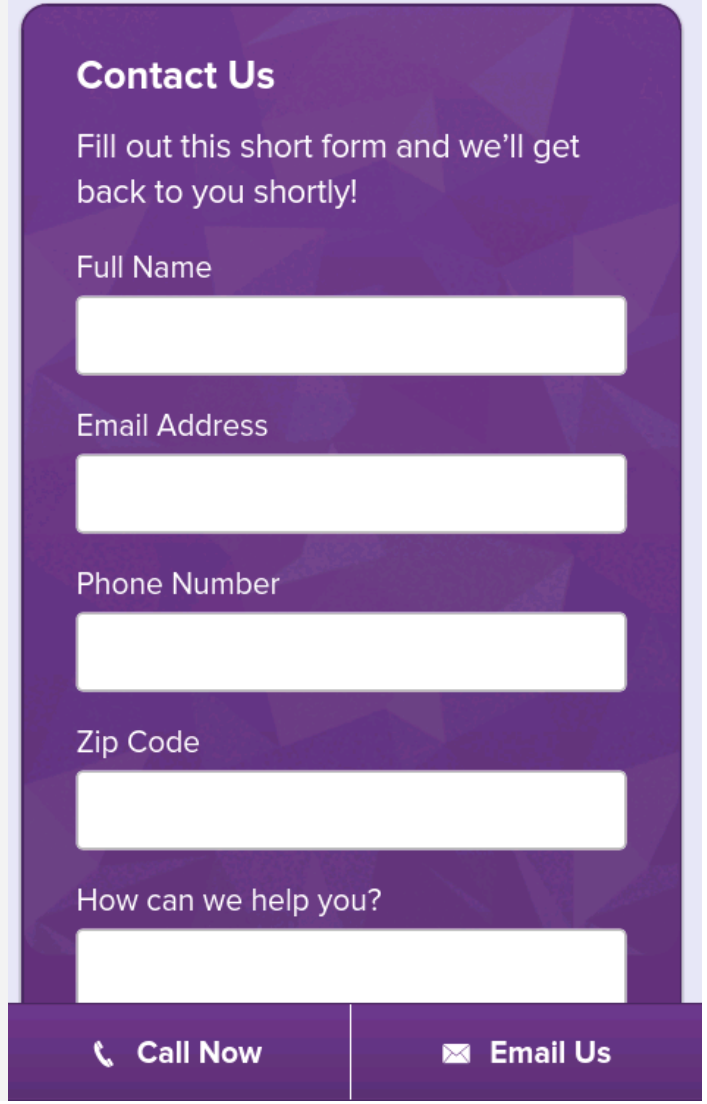
See the paper

Summary

Brahmastra enabled us to successfully analyze third-party code in 2.7x more apps than the state-of-the-art GUI testing tool:

(1) A study of apps with ad component shows that 80% kids apps displayed ads that attempt to collect personal info and 36% show inappropriate content

(2) A study of apps with Facebook Connect shows that 13 popular apps were still vulnerable to the access token attack



Questions?