Mining Sandboxes

A dissertation submitted towards the degree
Doctor of Engineering (Dr.-Ing)
of the Faculty of Mathematics and Computer Science
of Saarland University

by

Konrad Jamrozik

Saarbrücken, 2018

Day of Colloquium	22 / 10 / 2018
Dean of the Faculty	Univ.-Prof. Dr. Sebastian Hack
Chair of the Committee	Prof. Dr. Wolfgang Paul
Reporters	
First reviewer	Prof. Dr. Andreas Zeller
Second reviewer	Prof. Dr. Christian Rossow
Academic Assistant	Dr. Rahul Gopinath

Abstract

Modern software is ubiquitous, yet insecure. It has the potential to expose billions of humans to serious harm, up to and including losing fortunes and taking lives. Existing approaches for securing programs are either exceedingly hard and costly to apply, significantly decrease usability, or just don't work well enough against a determined attacker.

In this thesis we propose a new solution that significantly increases application security yet it is cheap, easy to deploy, and has minimal usability impact. We combine in a novel way the best of what existing techniques of test generation, dynamic program analysis and runtime enforcement have to offer: We introduce the concept of *sandbox mining*.

First, in a phase called *mining*, we use automatic test generation to discover application behavior. Second, we apply a *sandbox* to limit any behavior during normal usage to the one discovered during mining. Users of an application running in a mined sandbox are thus protected from the application suddenly changing its behavior, as compared to the one observed during automatic test generation. As a consequence, backdoors, advanced persistent threats and other kinds of attacks based on the passage of time become exceedingly hard to conduct covertly. They are either discovered in the secure mining phase, where they can do no damage, or are blocked altogether.

Mining is cheap because we leverage fully automated test generation to provide baseline behavior. Usability is not degraded: the sandbox runtime enforcement impact is negligible; the mined behavior is comprehensive and presented in a human readable format, thus any unexpected behavior changes are rare and easy to reason about. Our BOXMATE prototype for Android applications shows the approach is technically feasible, has an easy setup process, and is widely applicable to existing apps. Experiments conducted with BOXMATE show less than one hour is required to mine Android applications sandboxes, requiring few to no confirmations for frequently used functionality.

Zusammenfassung

Moderne Software ist allgegenwärtig und zeitgleich unsicher. Dies stellt ein Risiko dar, welches Milliarden Menschen verwundbar gegenüber Schadsoftware macht und dessen Folgen sich bis hin zu Vermögensverlust und Lebensgefahr ausweiten können. Gegenwärtige Ansätze zur Gewährleistung der Sicherheit in Computerprogrammen gestalten sich entweder höchst kompliziert und aufwendig, beeinflussen massiv die Benutzbarkeit oder aber stellen sich als nicht effektiv genug gegen resolute Angreifer heraus.

In dieser Arbeit präsentieren wir einen neuen Lösungsansatz, welcher die Sicherheit einer Applikation drastisch erhöht, zeitgleich sowohl kostengünstig als auch einfach einzusetzen ist und ferner nur minimalen Einfluss auf die Benutzbarkeit des Programmes nimmt. In einem neuartigen Verfahren kombinieren wir die Vorteile von etablierten Methoden der Testgenerierung, dynamischer Programmanalyse und kontrolliert restriktiver Laufzeitumgebung und stellen das Konzept des *Sandbox Mining* vor.

Im ersten Schritt verwenden wir automatische Testgenerierung in der *Mining* Phase, um das Verhalten der Applikation zu erkunden und zu beobachten. In einer weiteren Phase verwenden wir eine sogenannte *Sandbox*, um jegliches bisher nicht beobachtete Verhalten der Applikation während des normalen Betriebes zu unterbinden. Bei Nutzung einer Applikation in solch einer Sandbox sind Nutzer somit geschützt vor plötzlicher Änderung des Verhaltens der Applikation im Vergleich zu dem bereits beobachteten Verhalten während der Testgenerierung. Folglich sind Hintertüren, komplexe, persistente Bedrohungen sowie andere Angriffe, welche auf der Verzögerung ihrer Durchführung beruhen außerordentlich schwer umzusetzen, ohne dass diese dabei entdeckt werden. Diese Bedrohungen werden entweder während der abgesicherten Mining Phase, in welcher sie keinen Schaden anrichten können, entdeckt oder werden während der Ausführung in der Sandbox verhindert.

Der Mining-Prozess ist günstig in seiner Umsetzung, da das normale Verhal-

ten des Programmes vollkommen automatisch erlernt wird. Zur gleichen Zeit bleibt die Benutzbarkeit des Programmes unbeeinflusst und der Mehraufwand der Laufzeitabsicherung durch die Sandbox vernachlässigbar gering. Ferner ist das erlernte Verhalten verständlich und in einem von Menschen lesbaren Format aufbereitet; daher sind jegliche unvorhergesehenen Änderungen im Verhalten des Programmes selten und einfach zu erklären. Unser BOXMATE Prototyp für Android Applikationen zeigt, dass das Verfahren technisch realisierbar ist, einen einfachen Einrichtungsprozess bietet und weitflächig anwendbar auf bestehende Applikation ist. Bei der Durchführung von Versuchen mit BOXMATE hat sich gezeigt, dass es weniger als eine Stunde bedarf um Sandboxes für Android Applikation zu generieren und es derweil nur wenige oder gar keine Konfirmation der Regeln für die häufig genutzten Funktionen erfordert.

Acknowledgments

Working towards a PhD degree is a journey which cannot be undertaken alone. Let me thank my loving family for their unyielding support: My dad, my brother and my mom. I also want to express my deepest gratitude to the following:

Sascha Just, for translating the abstract to German.

Curd Becker, the most patient administrator under the sun.

Reviewers and the examination board, for taking the time to review my work.

Philipp von Styp-Rekowsky, who was the perfect collaborator and whose work and help was crucial for my research.

Marcel Böhme and Alessandra Gorla, for providing excellent close-range support when the inevitable hard times came, which helped me persist. Marcel was one of the best discussion partners when it comes to research, and actually, life in general.

Gajusz Chmiel and Dagmara Saganowska, my old friends who always provided long-distance support and advice when I needed it.

Adam Grycner, for being a world-class listener.

Savina Takeva, for her unmatched empathy and life wisdom I took to heart.

Rafaella Antonyan, for her unconstrained spirit and good-heartedness.

Marta Podgórska, for teaching me how to make pancakes, eat spaghetti properly and open toothpaste. Most importantly, for her patience when it comes to planning trips.

Mateusz Malinowski, who did his PhD at the same time as me. I believe Mateusz understood my internal monologue the most. We spent countless hours together, punching each other in the face (on boxing classes, don't worry), drinking tomato juice and cocoa (to the horror of waitresses) and talking about every topic imaginable, maybe except hamsters (we have to fix that).

All other friends. I will be forever grateful for our time spent together.

Finally, and most importantly, I want to thank my advisor, Andreas Zeller. Given a choice, I would not replace Andreas with any other mentor. Andreas had many virtues as a research guide, including excellent research ideas, unwavering conviction in them and the ability to convey it, which kept me motivated and passionate for so many years. Most importantly to me, I felt Andreas was adamant about fairness and treating his pupils well. I felt he cared for my success, which ultimately allowed me to complete this thesis.

Contents

Chapter 1

Introduction

In XXI century software is interwoven into civilization infrastructure. Software transfers your money between bank accounts, controls your house lights and temperature, manages power plants, sends texts messages from your phone, automates assembly plants and even self-drives cars. This is extremely convenient, but is it safe? Can you be sure your smartphone is not sending your private data to criminals? That your smarthouse is not secretly telling burglars you are away on vacation? That your banking application hasn't been infiltrated? That the keys you pressed to enter your passwords haven't been logged and sent off to the Internet?

You can't. But there is hope.

Researchers and other security experts spent decades working on methods that will make software convenient and easy to use, yet secure. While in principle we can build secure software, the costs and expertise required to build truly impenetrable programs is prohibitive. Secure software is a notoriously hard problem even for entities having vast resources at their disposal. Consider a series of high-profile data breaches from big companies, like LinkedIn [77], Adobe [78] or Tumblr [79], in each of which millions of users credentials have been stolen. Even in products with world-wide adoption there is a never ending stream of security holes [80], including the well-known Heartbleed bug [81], to name just a one high profile case.

This thesis introduces a new technique, *mining sandboxes*, that promises an approach to securing software that is both cheap and effective. We combine state-of-the art of existing program analysis and sandboxing techniques in a novel way that enables very effective, automated, highly understandable way

of preventing the applications you use from harming you, while not impacting usefulness of them in any noticeable way.

1.1 Publications and thesis structure

The remainder of this thesis describes in detail the patented [37] technique contributed by it, *mining sandboxes*. The idea was first published by Jamrozik, von Styp-Rekowsky and Zeller at ICSE 2016 [38] with additional implementation details given by Jamrozik and Zeller at MOBILESoft 2016 [39]. The structure of this thesis is as follows:

Chapter 2 provides background for this thesis contributions. First, we state the problem we aim to solve and possible challenges that need to be addressed while doing so. Next, we discuss existing solutions and their strong and weak points. Finally, we gain insights from the existing state-of-the-art approaches, leading to the invention of the *sandbox mining* concept.

Chapter 3 introduces the *sandbox mining* concept for securing software. We introduce the concept itself, as well as important principles and properties arising from it. We show how, in principle, sandbox mining is superior to existing approaches. Finally, we describe possible and predicted offsets that have to be made when implementing sandbox mining as a technique, thus giving rise to research questions guiding the rest of the work done for this thesis.

Chapter 4 describes the algorithmic and technical details of our input generation implementation, which is a core component of sandbox mining. We describe the input generator we implemented, DROIDMATE. We discuss DROIDMATE features as compared with other input generators, justifying our choice to create it in the first place. We discuss in detail DROIDMATE design and architecture, including execution phases, components (like exploration algorithm, GUI automation, and Android framework API calls monitoring), modules and resources. Lastly, we list DROIDMATE limitations.

Chapter 5 treats about BOXMATE, our extension of DROIDMATE that enables us to infer sandbox rules and enforce these rules on Android applications. This is another critical component of sandbox mining. First, we introduce the Android permission system and point out its flaws. Next, we

discuss the two policies for enforcing behavior we implemented, *API call enforcement policy* and *event-bound API call enforcement policy*. We provide a SNAPCHAT Android application case study showing an example of API call enforcement policy definition. Finally, we discuss two possible implementations of a sandbox.

Chapter 6 discusses the experiments we conducted to answer the first three questions posed in Chapter 3. We discuss the experimental setup and the obtained results, answering the research questions.

Chapter 7 is a second study we conducted to answer the final, fourth research question. It has similar structure to the previous section.

Chapter 8 summarizes our work, concludes and describes possible future directions of research opened up by concepts introduced in this thesis.

1.2 About glossary

While reading this dissertation please note there is a Glossary. It clarifies potentially confusing terms, abbreviations and how they relate to each other.

Chapter 2

Background

2.1 Secure and malicious software

We say *software is secure* when it is impossible for it to behave in a *malicious* way, even when influenced by an adversarial hacker. A program can be malicious by design, or because it has security vulnerabilities which can be exploited to make it malicious, or just because it has defects that result in malicious behavior.

It might seem trivial to determine if software is malicious, but is it? Let's say your banking application transferred some money from your account. If this is a malicious action depends entirely on the fact if you initiated the transfer, or if it was made without your knowledge. If you ordered the transfer because you were buying a power-up in a game, all is fine. But if the transfer occurred suddenly in the background, without your knowledge, to a bank account you don't know, this action is malicious.

The example above illustrates that for software to be considered malicious, it has to drain some of your *valuable resources* and do so in an *unexpected way*. By *valuable resource* we mean anything that you might wish to spend consciously for some gain, but otherwise should not be spent, as it is valuable to you. Some examples include money, privacy, reputation, attention or health. Examples of spending these resources in an intended way include paying for some service, sharing personal information for identity verification or undergoing medical X-ray. Malicious scenarios include an application sending messages to premium numbers without your knowledge, application covertly logging and sending your bank account passwords to hackers, or self-driving cars hacked to speed up

instead of slowing down when brakes are pressed.

2.2 Problem statement and challenges

Sandbox mining, the technique contributed by this thesis, solves the problem of *increasing software security* in a way that is *useful in practice* and *widely applicable*.

To make software more secure one needs to be able to *know how it behaves*, determine *which of the behavior is malicious* and *ensure malicious behavior is blocked*. Usefulness in practice means the application of the solution cannot cripple or otherwise limit the secured software in unacceptable ways. Thus, the usability and functionality of the secured software cannot suffer significantly. Consequently, the solution has to be very *automated* and it needs to ensure that *intended behavior is allowed*. Wide applicability means it has to work on existing software, without too expensive and complicated setup process.

Following sections give an overview of existing solutions for securing software and how they fare against our requirements. We limit the overview to methods working on existing software, as this is one of our prerequisites. Thus, any methods that base their security on the fact the software is constructed in a specific way or adapted to some security framework are not discussed. They would require writing new or rewriting significant portions of the programs in question. Because our proof-of-concept prototype of the mining sandboxes technique is implemented on Android (see Chapter 4 and Chapter 5), we pay additional attention to related work pertaining to Android.

2.3 Manual methods of securing software

The most obvious way to ensure a program is safe to use is to conduct a *security audit* and fix any problems found. Audits however don't scale, as they require human expertise and thus are expensive. From the point of view of an end-user another way of ensuring security is to use only *certified software*. This however is just delegation of the problem: Now the certifying authority needs to check the software is secure. In addition, the user has to trust the authority and the certificate, which just adds a level of indirection that can contain potential security vulnerabilities. Another manual method is to write by hand a *formal specification* and check (possibly automatically) that the software indeed adheres to it. Unfortunately, like with security audits, writing specifications is very time

consuming task requiring expertise.

All of the above given methods aim at guaranteeing the software itself is going to be safe, and before it is actually run. There is an alternate way: Provide a *set of behavior rules*, also called a *behavior ruleset*, which the secured program behavior is forced to obey. Mechanisms for enforcing such rules are described in Section 2.4. First, however, let's discuss one way of enforcing a behavior ruleset, called *user-controlled policies*.

2.3.1 User-controlled policies

One kind of enforceable policies are *user-controlled policies*. That is, the policy rules are provided automatically or manually, by experts. It is left to discretion of the end-user if the rules should be enforced. Prime example of this is *User Account Control* dialog [82] in Windows family of operating systems. Every time a user is going to do some potentially malicious operation, like install a program, enable executable macros in a spreadsheet, or give a program administrator rights, she is presented with a dialog box asking for confirmation. Mobile operating systems like Android also have the same concept. For example, on Android, since version 6, the first time given Android app wants to call some phone number, user is presented with a dialog box asking to give the application permanent access to all permissions from the PHONE permission group, including the CALL_PHONE permission [83].

User-controlled policies invariably are either *too broad* or *too specific*. Consider an example of an application asking the user if it can make phone calls. When allowed, the application will be able to make *any* phone calls, to *any* numbers, at *any* time, even when the user is completely unaware of it. As another example, if the user allows application to read her contacts, it will be also granted the right to write and delete the contacts [83]. These cases pose a huge security risk: If the applications in question were actually malicious, they could deplete the users account by making phone calls or delete all of her contacts without her permission.

The problem here is the policy is too broad. If, on the other hand, the policy would be too specific, it might end up infuriating the user. Imagine an application asking user to confirm every single phone call she wants to make, or every photo she wants to take. The application would be considered unusable. There are more problems plaguing user-controlled policies. Often the question asked by the dialog box doesn't make sense to the reader. In addition, usually it is an *all or nothing* proposition: Allow the program to do everything, including deleting all your data (if it is not malicious, it won't do it, but how can you

know?), or you cannot use the program at all.

The permissions required by the application (which end-user will have to confirm when using the app) have often to be determined by developers, as it is the case with Android platform [84]. If the developer makes a mistake by not predicting that given permission might be required by the application, it will crash. If, on the other hand, the developer gives an application too much permissions, it might inadvertently become malicious. A study by Felt et al. [24] has shown that over 33% of investigated Android applications are overprivileged due to their developers being unsure which permissions are actually required.

2.4 Runtime enforcement

Many malicious behavior prevention mechanisms can be categorized as *runtime enforcement* schemes. They *enforce during runtime* (i.e. during actual program usage by end-users) a set of rules, to ensure no malicious behavior occurs. Specifics of the mechanism and nature of the rules enforced depend on the platform. We will now discuss most important scenarios.

A user might wish to protect herself with anti-virus software that checks all downloaded data against *malware signatures*. The malware signature databases are maintained and continuously updated at significant effort by companies with plenty of resources. The crippling flaw of this solution is that it doesn't work very well against new malware. Such malware is unknown at the moment of appearance, thus no anti-virus software has patterns for detecting its signature. Moreover, more advanced malware can evade detection by employing many evasion techniques [96], including advanced methods like metamorphic code [92].

Network communication can be monitored by *firewalls* [93], *intrusion detection systems* [95] and *deep packet inspection* [94]. Firewalls use sets of static rules to determine which network traffic is allowed based on the protocol (`"http"`, `"ftp"`, etc.), host, port and other properties. While this filters out a lot of unwanted traffic, determined attacker can still find a way to send a network packet allowed by the static rules, e.g. by gaining access to one of the allowed hosts. One way to combat this is to complement firewalls with *intrusion detection systems* (henceforth IDSes), introduced by Denning [21] in 1987.

IDSes work on the principle of *anomaly detection*, which is simple: Anything not within the bounds of "normal" behavior is an anomaly and so is potentially malicious. The most important distinction from the techniques discussed so far is that the "normal" behavior doesn't have to be described by manually provided set of rules, but can be inferred *automatically* in form of *models*. Details of this

process are described in Subsection 2.5.3.

Sometimes there are no traffic anomalies because the malicious payloads are well hidden within the usual incoming messages. In principle deep packet inspection could find such payloads, but it suffers from similar problems as malware detectors. In addition, there are major concerns about deep packet inspection being used to violate privacy and reduce Internet openness.

Operating systems usually encase every applications running on them in a *sandbox*. Such encased application has high degree of isolation from its environment and other applications. It can still interact with the rest of the system, but many of such interactions are monitored and governed by a set of rules. Sandboxing is described in more detail in Subsection 2.4.1.

2.4.1 Sandboxing

Nontrivial programs communicate with the surrounding environment: The file system, the network, processes of other programs (using pipes, shared memory, etc.) or other OS subsystems. Often the communication is happening through unified operating system API, as it is the case with Android [97]. Allowing any program to access anything and anytime could wreak havoc. Imagine a program erasing contents of your entire file system due to an erroneous implementation. To prevent such or any other malicious behavior from happening, the operating systems have mechanisms for enforcing the *principle of least privilege* [55], introduced by Saltzer and and Schroeder in 1975. This principle simply states that given program should be granted the absolute minimum necessary privileges for it to fulfill its purpose, and nothing more. In context of operating systems and applications running on them, this principle can be realized by *sandboxing* the application, i.e. by ensuring all of its communications with other subsystems and processes are monitored and if necessary, blocked.

On Android, applications can communicate with external world only through the Android framework API Android [97]. The sandboxing is realized by checking any calls made to this API against a set of permissions. These permissions form a basis of *user-controlled policies* as described in previous Subsection 2.3.1 and thus they suffer from the same problems. Ultimately, this means that even though the principle of least privilege is implemented in Android, the protection it offers is not nearly enough to ensure that the Android users are secure while also maintaining functionality of the sandboxed programs.

2.5 Automated behavior discovery

Determining program behavior manually is hard, as programs are complex. Thus, an automated approach is in order. At least partially automated ways to discover program behavior are collectively referred to as *program analysis* methods. Any program analysis method falls into at least one of the two categories: *static analysis* and *dynamic analysis*. Static analysis [51] allows us to reason about a program without actually executing it. Static analysis extracts information from program code (binary code, byte code or source code) and any other available artifacts, like GUI layout files, localized strings, etc. Dynamic analysis [22], on the other hand, actually executes the analyzed program to observe its behavior. Dynamic analysis is often used to *model behavior* of single programs, or to model behavior of entire systems to *detect anomalies*, as described in Subsection 2.5.3.

2.5.1 Static analysis

The distinguishing advantage of static analysis is the fact the analyzed program doesn't have to be executed, which makes such analysis easily applicable as compared with dynamic analysis approaches. This results in various static analysis methods being widespread in practice.

SPIN is a model checker [36] for embedded systems. Microsoft's SLAM [11] checks if C programs use APIs correctly. These are just two tools representing a widely used class of tools statically checking if software obeys given *software model*. Another kind of static analysis tracks the flow of sensitive data within the programs. They can detect situations like programs reading private contacts and sending them to unknown (presumably malicious) servers. Prime examples of such tools for Android are CHEX [45] and FLOWDROID [6].

Important static analysis technique is *symbolic execution* introduced in 1975 in the SELECT system by Boyer et al. [16]. By building symbolic expressions over constraints expressed in the source code, symbolic execution enables reasoning about which parts of program would be executed given specific inputs. This enables static analysis to gain confidence about how program would behave when executed, without actually running it. Nowadays powerful tools leveraging both static and dynamic analysis exist, as described in Subsection 2.5.2.

Limitations

Even with its widespread use and advantages, all static analysis techniques suffer from many limitations, as listed below. In security context, malware writers use these limitations with impunity to exacerbate the problem of understanding program behavior statically.

Program complexity Software systems can be composed of dozens of software libraries, frameworks and components, making the code line count run well into millions. Even modern static analysis tools running on powerful machines cannot reason about such programs in reasonable amount of time. To circumvent this problem, static analysis is either applied to single components, or provides abstractions over units of analyzed systems. In the first case, we run into the limitation of **unknown environment**. In the second case, because we cannot abstract over the components precisely, the problem of **overapproximation** arises.

Unknown environment When the analysis is limited in scope to given software unit, everything beyond it is treated as opaque environment. Consider analysis of a program which communicates with a database, file system, network or other programs. Because these are systems and resources external to the analyzed programs, we cannot statically determine their exact behavior. The best we can do is to approximate it, which unfortunately results in **overapproximation**.

Overapproximation Static analysis is inherently imprecise, because of **unavailable runtime values** and fundamental limitation of **undecidability**. That is, we have to assume more can happen that will ever actually happen, otherwise we risk not considering a malicious behavior. In security context this means static analysis often raises alarm of potential vulnerability while there is no actual threat. This, in turn, burdens human experts who have to investigate such false alarms manually, wasting time.

Unavailable runtime values Given **unknown environment**, for example an external database being used by the analyzed program, we cannot know what data exactly it contains. We thus need to assume all possible data, **overapproximating**. We encounter the same problem if the analyzed program reads contents of the file system, or reads data arriving through network.

In some cases, like e.g. reading a configuration file from the file system, we could try to extend our analysis to read the file contents. However, this assumes we have access to the file system in which the analyzed program would run and that the file won't change during runtime. Otherwise we would be entering into the realm of dynamic analysis. In other cases, like data transferred through network, we have no options, as the data will be available only at runtime.

Runtime values can be even entire programs. Consider an application that upon start downloads from a server another program, installs it and runs it. It might be the case that any malicious activity is done only by the downloaded program, which is inaccessible to static analysis. Unfortunately, we cannot mark such scenarios as being obviously malicious, as downloading entire programs or program modifications is a valid use case for plugins, feature updates and security patches.

Undecidability As a consequence of the halting problem we know that static analysis is undecidable. Consider a program that reads a string representing another program and runs it. We do not know the value of the string, so it can be arbitrary program. Thus, due to the halting problem, we cannot know if it will halt or not. The undecidability of static analysis has a far reaching consequence: Not only current tools have to overapproximate behavior; in all generality they will always have to do so.

Information erasure in object code A significant drawback of static analysis applied in adversarial security setting is the fact the source code is unavailable. It is the easiest to analyze the original source code, but if we suspect the code is malicious, chances are we don't have access to the source code in the first place. This leaves us with bytecode (as with e.g. .NET, Java Virtual Machine or Android) if we are lucky, or just machine code. Such representations of the source code underwent multiple transformations by the compiler and other tools, including optimizations, **obfuscation** and **encryption**, making it much harder to reason about, sometimes even impossible. Any false alarms resulting from overapproximation by static analysis on such low-level code would be exceedingly hard to check manually due to the loss of human-readable high-level structure of the original source code.

Obfuscation If program complexity wasn't enough of a problem, the analyzed program code (source code or object code) is often additionally **obfuscated**. Obfuscation scrambles the available symbols, adds bogus control flow state-

ments and so on, ultimately making programs even orders of magnitude more complex, and thus harder to analyze. Unfortunately, obfuscation cannot be just immediately deemed malicious, as it is also used to protect intellectual property. Android tool chain, for example, has a built-in obfuscator, ProGuard [99].

Encryption Similarly to obfuscation, **code encryption** is a technique used to protect intellectual property. This fact is exploited by attackers. If the analyzed code is encrypted, any analysis is impossible as it is just scrambled data. The encrypted code has to be decrypted before it is executed, but then reasoning about it would require actually running it, which is beyond the scope of pure static analysis.

2.5.2 Dynamic analysis

Dynamic analysis [23], as opposed to static analysis, does not analyze static artifacts, but instead analyzes data obtained from executing the program. To conduct dynamic analysis one needs to first *generate inputs* (manually or automatically) and feed them to the program, resulting in a set of *executions*. One has to observe the program during the executions to gather relevant data to be further analyzed, e.g. by observing calls to OS API methods.

Input generation

Critical challenge for dynamic analysis that is not present in static analysis is the problem of *input generation*. To observe data obtained from a running program, one has to execute it. To execute a program, one has to provide inputs for it. To cover majority of all possible behaviors of given program, one has to execute it many times, each time with inputs differing enough as to make the program behave in a different way. Thus we arrive at the problem of providing a large set of heterogeneous inputs for a program. The inputs may need to be varied and arbitrarily complex: Configuration files, images, sequences of GUI clicks, command line commands, voice commands, entire programs (e.g. used as inputs to compilers), natural language files (e.g. for translator software), highly structured formats like XML files adhering to given schema, and so on. The inputs can be inputted from many locations, like file system, database, network or GUI I/O events, like mouse clicks or swipes. Furthermore, to elicit some specific behaviors, the inputs might need to be provided in appropriate sequence. All of these considerations make input generation a hard problem.

Input generation methods can be split into two classes: *system-level*, with the examples of such input types given above, and *unit-level*, which do not treat the program as a black-box, but inject inputs directly into its internals, e.g. by constructing objects (in programming languages supporting such constructs) and making specific sequence of method calls with generated arguments. Major disadvantage of unit-level input generation is that it requires knowledge of the code (source or object), thus it suffers from the same flaws as static analysis in that regard. If the code is complex, obfuscated or encrypted, it is hard to impossible to generate meaningful unit-level inputs. Furthermore, unit-level input generation has more application in regression testing than for discovering high-level program behavior. Thus, **in further discussion, we always assume system-level inputs, unless noted otherwise**. Overview of existing methods for input generation is given in Section 2.5.2.

Limitations and comparison with static analysis

Dynamic analysis solves or significantly mitigates a lot of static analysis problems listed in Subsection 2.5.1. However, it suffers from its own problems, on top of the challenge of input generation. Let us now describe dynamic analysis strengths and flaws, and compare them with static analysis.

Program complexity Often, the more complex the program, the harder it is to *generate inputs* for its execution. Fortunately, most of the time relatively simple inputs can cover majority of the primary use cases of given program. More complex programs are also more likely to suffer from **execution performance issues**, i.e. run longer, thus elongating the entire process of applying dynamic analysis.

Known environment During dynamic analysis the environment is known, in stark contrast to static analysis. As an example, recall that static analysis cannot make any assumptions about the file system in which the program will be run. For dynamic analysis the file system (or its mocked implementation) has to be present for the program to work. This enables dynamic analysis to work with concrete values where static analysis could only assume that anything can happen. Static analysis suffered from overapproximation, but dynamic analysis suffers from **underapproximation**.

Underapproximation Like static, dynamic analysis is also inherently imprecise, but for a different reason. While static analysis is often forced to assume

that given variable can have any value, dynamic analysis can only observe small subset of all possible values of that variable. Namely, the values observed from the executions obtained using the *generated inputs*. This leads to **underapproximation**: Dynamic analysis will overlook many possible values and thus, will not conduct executions leading to undiscovered behaviors. This means the program behavior observed during dynamic analysis will never be complete, except for the most trivial of programs. Adversarial hackers can use this to their advantage by employing **evasion** techniques, including **time bombs**.

Execution performance Huge programs might be very slow to execute. Some of dynamic analysis techniques avoid this problem by summarizing some parts of the executed program. Namely, instead of running them to obtain the values, they just treat these parts as **known environment** and return concrete values. The values might have been obtained in the first place by remembering them after one-time slow execution of the summarized parts. While this mitigates the performance problem, of course it leads to increased risks of **underapproximation**, as we operate under the assumption the summarized values do not change, which might not be true.

Available runtime values Because the **environment is known**, the runtime values are available, be it files in a file system, data in external database, network packets or anything else. However, we are of course limited to observing only the values that are currently available, either as test fixtures or real-world data. All values processed by a program can be considered its input, and thus providing appropriate values is an *input generation* problem.

Incompleteness Dynamic analysis doesn't suffer from *undecidability*, but it is *incomplete*. In practice, the observed executions will allow us to reason only about a subset of all possible behaviors of analyzed application, thus we are always at risk of missing important and/or malicious behaviors.

Source code not required For dynamic analysis it doesn't matter which kind of code is available. As long as the program is executable, dynamic analysis has no benefits from the presence of the original source code or object code.

Obfuscation While obfuscation can significantly hinder performance of static analysis, it has negligible effect on dynamic analysis, if any. If the data observed

during executions contains obfuscated code symbols, the data will be harder to reason about manually. Fortunately, this problem occurs rarely.

Encryption Encryption can completely defeat static analysis, yet it has no influence on dynamic analysis. Dynamic analysis just executes code, so as long as the encrypted code gets decrypted at runtime or is otherwise executable, there is no downside, except for a possible performance hit of decrypting code.

Evasion **Underapproximation**, and in all generality **incompleteness** of dynamic analysis, has far reaching consequences in security context. A skillfully crafted malware might ensure that all behaviors observed during dynamic analysis are benign, while the malicious actions are triggered only in actual usage by the end-user, where they can do harmful actions, being previously undetected. One broad category of evasion techniques are **time bombs**.

Time bombs Malicious programs might purposefully delay malicious actions to prevent detection. Consider a program that activates itself only on a specific day and time, 3 months from now. No amount of dynamic analysis will be able to detect that, unless the program is analyzed at the exactly right time. Of course, one could try to use manual or static analysis to find the time bomb, but then one has to deal with all of the disadvantages of these techniques.

Existing approaches

The defining feature of the existing dynamic analysis tools is how they generate inputs, as this determines the tool's capability to discover behavior, a core trait determining dynamic analysis power. Because the generated inputs enable program executions which then can be codified as tests, input generation is often called *test generation*. In this thesis we use the terms *input generation* and *test generation* interchangeably.

Purely random testing The simplest way is to generate inputs in a purely random fashion. This falls under the broad category of *random testing* [33]. Tools working like that are often called *monkeys* and are usually bundled with software development kits of popular developer platforms. One example is the Monkey tool [100] in Android SDK, which randomizes screen coordinates for each click, touch or gesture on an application GUI, being completely oblivious to anything except the screen size.

Adaptive random testing More advanced random input generation tools leverage data observed from already conducted executions to guide the process of input generation. *Feedback-directed Random Test Generation* introduced by Pacheco et al. [52] is a unit-level input generation technique that uses the execution result to determine if the provided input was redundant, illegal or otherwise dispensable. It leverages this information to construct further, useful inputs.

The same principle but on a system-level is applied by the Dynodroid tool by MacHiry et al. [46]. Dynodroid randomly generates a set of UI events (like clicks, drags or text inputs) and system events (like PACKAGE_ADDED or TIMEZONE_CHANGED). Unlike Monkey, Dynodroid obtains feedback from its actions in form of the currently displayed GUI layout, so it is aware which GUI elements can receive which kinds of events (buttons can be clicked, text input fields can have text entered, etc.) and how many times these elements have been already interacted with. It uses this information as a feedback guiding its random actions to the GUI elements which haven't been exercised yet. This results in a biased random strategy that prioritizes randomizing inputs that discover yet unseen behavior.

Grammar-based fuzz testing It is a kind of random testing that ensures the randomly generated inputs adhere to a given *grammar*. This is necessary when generating highly structured inputs. Consider a task of testing a compiler. Compiler takes as input a string representing an entire program. If the program will not be syntactically correct, the compiler execution won't go beyond syntax check, thus the observed behaviors will be severely limited. To pass the syntax check, the generated inputs, while random, should be syntactically valid programs.

A prime example of a tool employing such technique developed by Holler et al. is LANGFUZZ [35]. It takes as input a context-free grammar of given language as well as code fragments of that language. Next, it recombines the code fragments creating new random fragments, but adhering to the grammar. LANGFUZZ has proven to be successful in finding security vulnerabilities in JavaScript interpreter. It found 105 serious vulnerabilities within three months of operation, worth in total 50.000$ in bug bounties [35].

Search-based testing A plethora of *search algorithms* like hill climbing or genetic algorithms can be used to generate inputs. Many search-based tools exists, as described in a survey done by Anand et al. [4]. Prominent example is EVOSUITE, a tool by Fraser and Arcuri [27]. EVOSUITE is a unit-level search-

based test generator for Java programs. It uses a genetic algorithm to evolve a population of Java test suites. EXSYST by Gross et al. [32] adapts EVOSUITE genetic algorithm to work on GUI actions sequences instead of unit test suites. A more recent example in the Android domain is the pareto-optimal multi-objective search-based system-level approach of SAPIENZ by Mao et al. [48].

Systematic input generation Böhme and Soumya have proven that generating inputs randomly is surprisingly efficient at covering enough behavior for all but the most safety-critical software [15]. However, sometimes we require to discover behaviors that are triggered by very specific, complex input configurations. Random input generation is too unlikely to discover them. Instead, we can leverage *systematic input generation* also known as *systematic test generation*. Instead of randomizing, systematic test generation methodically explores the possible inputs space, often with the aid of a *model* and information extracted from the program by means of static analysis. Examples of systematic input generation approaches follow.

Model-based testing The inputs can be generated in a way that checks if the program adheres to a given *model*. A^{3}E tool by Azim and Neamtiu [8] uses static analysis on Android app's bytecode to build the app's GUI screen transition model. The generated inputs are sequences of clicks on the GUI. The tool aims to generate minimal sequences required to cover as many GUI screens as possible, as defined by the extracted GUI model. Major drawback of this approach is that the model is extracted statically, so the process suffers from majority of the flaws of static analysis. To mitigate this, one can extract model completely dynamically, as it is done by SwiftHand by Choi et al. [18]. SwiftHand extracts GUI model purely dynamically, by observing the current GUI state, and uses machine learning to fix any model inconsistencies on the fly.

CRAWLJAX by Mesbah et al. [49] and WEBMATE by Dallmeier et al. [20] are model-based tools for exploring behavior of Web applications. Web domain poses additional challenge of server-side of explored app being usually unavailable. In addition, the client-side is spread across many languages and formats like HTML, DOM, CSS and JavaScript.

We listed some GUI-model-based input generation tools. Yet the field is much richer, boasting various approaches to build models, including axiomatic approaches, finite state machine-based models and labeled transition systems [4].

Dynamic symbolic execution Symbolic execution is a static analysis technique that determines which inputs will lead to execution of given part of program code. Symbolic execution suffers from all the usual problems of static analysis, with the unavailability of environment and thus concrete runtime values being especially pronounced. An extension of symbolic execution, *dynamic symbolic execution* (or DSE for short), first introduced by Godefroid et al. [29], actually runs the program, observing and using concrete values obtained from the program's environment.

A prime example of a DSE tool is SAGE by Godefroid et al. [30]. SAGE determines which constraints inputs have to obey to execute instructions in binary code. It then systematically negates the constraints, forcing different control flow, thus thoroughly exploring given program's behavior. SAGE has been applied to huge programs like Excel and has saved Microsoft millions of dollars in found bugs [30]. PEX is another tool from Microsoft by Tillman and Halleux [58] which uses the principles of DSE. PEX, however, works on unit level, automatically generating test suites covering code of given object class.

2.5.3 Modelling behavior for anomaly detection

We can observe programs and entire systems to reason about their behavior. Such observations are usually codified, again, in *models*. If we operate under the assumption our observations happened on benign behavior, we can use them for the basis of *anomaly detection*. Indeed, the first intrusion detection system introduced by Denning [21] detected intrusions by "abnormal patterns of system usage". This IDS *automatically learned* a statistical *model* from the logs of observed network traffic. Next, future network traffic was compared against this model and significant enough statistical deviations were considered to be potentially malicious. This statistical model was first case of detecting anomalies by comparison with an *inferred model*. To avoid the unacceptable cost of manually determining allowed behavior, later on a plethora of machine learning approaches had been applied to monitoring network traffic. Survey by Garcia-Teodoro et al. [28] discusses methods like Bayesian Networks, Markov Models, Neural Networks, Fuzzy Logic, and Genetic Algorithms.

Checking at runtime if any deviations from given model are present also has been applied within the scope of a single process. Forrest et al. [26] introduced in 1996 modelling of combinations of system calls to UNIX process. Any yet unseen combination was considered an anomaly. Later anomaly detection methods operating within a single machine (or *host*) target all kinds of platforms, including Web [40], Android [14] or Windows [42]. Such approaches monitor not only

system calls, but also file system and registry accesses, among others.

All of the discussed modelling techniques are a kind of dynamic analysis, so they suffer from the same problem: Incompleteness. If a new behavior is seen, is it good or bad? Detect too many anomalies, and the model is unusable, as each anomaly needs to be manually assessed. Let one malicious behavior slip without raising an alarm and you are in trouble. On top of that, because most of the advanced models are derived with machine learning methods which produce complex models, they are opaque to humans. Even if there is an effort made to present the inferred models in a human-readable way, they are still very cumbersome to analyze. Consider a set of thousands of sequences of low-level calls to a low-level operating system API methods. It is exceedingly hard to discern the actual meaning of such calls, and thus, hard to determine if it is a potentially malicious anomaly. As a consequence, it is practically impossible for humans to tailor the model to minimize the amount of spurious anomalies, while still detecting all the true (i.e. malicious) violations.

To make matters worse, in practice the models for detecting anomalies are learned from actual executions, as copious amount of data is required to derive models of sufficient quality. This causes a chicken and egg problem: To detect anomalies, and thus protect users, one has to train a model. To train a model, one has to observe actual user behavior using the system to be protected in the first place. While the model is trained, the system is vulnerable, because no model of it exists yet. Not only the users are vulnerable, but also because some of the observed behavior might be actually malicious, the learned model might be tainted, categorizing such behavior as benign.

All the listed flaws result in severely limited practicality of anomaly detection systems based on machine learning. As Sommer [57] summarizes:

> "Despite extensive academic research one finds a striking gap in terms of actual deployments of such systems: Compared with other intrusion detection approaches, machine learning is rarely employed in operational 'real world' settings."

2.6 Insights leading to a new approach

In this chapter we discussed how state-of-the-art methods for securing software fare against our requirements of increasing software security in useful, widely applicable way. Manual methods are too expensive to be widely applicable, thus we have to use automation. To prevent malicious behavior from happening

we have to be able to automatically reason about and enforce program behavior. We cannot do it statically, because it is too easy to prevent static analysis from reasoning about given program (Section 2.5.1). We could discover program behavior dynamically (Subsection 2.5.2), but then we need a good way to generate inputs, and ultimately we will never cover entire behavior due to dynamic analysis incompleteness (see the limitations in Subsection 2.5.2).

First key insight we have is that incompleteness of discovered behavior is not the primary problem, as we fundamentally want to *prevent malicious behavior*. We are more concerned with allowing malicious behavior than with blocking benign one. We thus can just consider the undiscovered behavior to be potentially malicious and take appropriate action, up to and including outright blocking it. Yet we do not want to block too much benign behavior, otherwise the secured application will become unusable and thus our solution won't be useful. We arrive here at a second key insight, that we already have good enough algorithms for automatic input generation (see Section 2.5.2) to discover enough behavior to not degrade applications usefulness in a significant way, while blocking malicious behavior. With these insights, we are ready to introduce the *sandbox mining concept* in Chapter 3.

Chapter 3

Sandbox mining concept

3.1 Introduction

Sandbox mining is a core idea for securing software contributed by this thesis, first described in [38] and patented by [37]. Sandbox mining, as illustrated on Figure 3.1, secures programs by *automatically finding sandbox rules* and encasing the programs in a sandbox obeying these rules. In essence, it is a combination of *dynamic analysis*, to *automatically* find the rules, and *runtime enforcement*, to ensure the rules are obeyed in production. The phase in which dynamic analysis is applied is called *mining*, as the sandbox rules are *mined* (found, discovered, extracted) during it. More precisely, the phases work as follows:

Mining During the *mining* phase, inputs are automatically generated using a *test or input generator*[1] and applied on the *program being secured* while it is monitored for *resource accesses* it makes, like OS API calls. This process is called *exploration* and results in *exploration log*. From this log, *mined behavior (specification)* of the program being secured is obtained, which is a set of observations about the resource accesses made.

Sandboxing Sandboxing phase is applied during actual application usage: The mined behavior is codified in a *behavior enforcement policy* enforced by a *sandbox*. If the application behaves in a way that is not allowed by the

[1] *Test* and *input* generators are the same for our purposes and can be used interchangeably. For more, see Glossary.

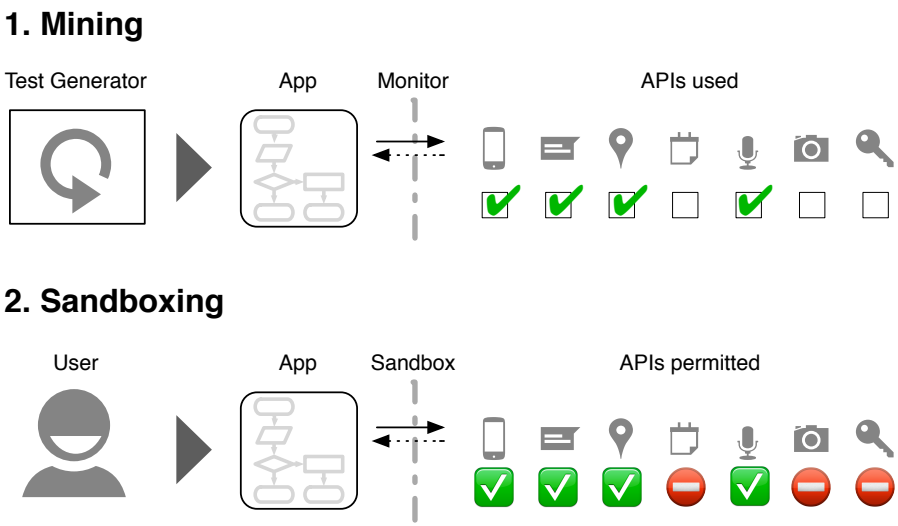

Figure 3.1: Sandbox mining concept: In the first phase inputs generated with a test generator are used to execute the application being mined, which allows us to observe its overt behavior. In the second phase this mined behavior is codified in a policy enforced by a sandbox in which the program is encased during actual application usage. This prevents unexpected and/or covert behavior. The figure shows a case in which the application didn't try to use API methods for accessing notes, camera and passwords during mining, and thus won't be able to do so without user's knowledge during normal usage.

policy, the behavior will be *flagged*, as it will be considered to be *unexpected and/or covert* behavior that *violates* the sandbox. Flagging will result either in an immediate block, or will require manual confirmation by the end-user, depending on the violation severity.

3.1.1 A concrete example

Consider a concrete instantiation of sandbox mining concept. We mine Android SNAPCHAT application (app, for short) for behavior pertaining to Android framework API calls (API calls, for short) and GUI events that trigger

them. SNAPCHAT behavior can be automatically explored by systematically clicking on its GUI with a test generator. The API calls can be contextualized to GUI events that triggered their call, like button clicks. When the test generator clicks the "login" button, we can observe that SNAPCHAT calls the following Android framework API method:

```
android.hardware.Camera.open(int)
```

This happens because after login SNAPCHAT immediately opens up feed from camera, so the user can start taking snaps. Based on that monitored data we could have mined following behavior: *Camera is opened after the "login" button has been pressed.* Naturally, after the test generator presses other buttons leading to the same screen with camera feed, we will mine other behaviors, relating presses of different buttons to open the camera. Such mined behaviors can then be codified in a *behavior enforcement policy* enforced by a sandbox. In the context of this thesis, we also *call behavior enforcement policy* a *behavior policy*, *enforcement policy*, *mined policy*, *sandbox policy*, *security policy* or even just *policy*. In essence, the policy is composed of *behavior rules* that have to be obeyed by the app. Such rules can then be reviewed by the user, before she starts using the app.

The SNAPCHAT in question could actually be a malicious variant pretending to be a genuine SNAPCHAT. Such app could, at any time, covertly take pictures with our phone and send them to hacker before we realize what is happening. But with the sandbox policy enforcing mined behaviors it won't be possible, as the camera usage will be limited to the cases in which the users clicks a button after which she fully expects for the camera to be activated, preventing its misuse.

3.1.2 Concept generality

Sandbox mining is a general concept, in which the *program being secured*, the *generated inputs*, the *monitored resource accesses* and the *behavior enforcement policies* can be defined, scoped and implemented in various ways.

The program being secured can be a desktop application, an entire web server, a mobile application or a "thing" in the Internet of things, like a centrifuge for separating nuclear material [41] or a self-driving car [76]. The interaction of the program with various resources can be monitored, including file system I/O, network communication, OS framework API calls, GUI gestures, factory robot arm sensors and actuators I/O, and remote commands to drones, to name a

few. The inputs can be generated using any system interface to the secured program: Through GUI, API or any other available means.

The enforced behavior policy is based on mined behavior, and thus it has to pertain to the monitored resource accesses. How policy violations in the sandboxing phase should be handled? In one scenario, accesses to critical OS APIs can be completely forbidden unless observed during mining, like ability to format entire file system. When brakes are pressed, software of a self-driving car might be allowed to brake, but not speed up. In more benign and unclear scenarios, the mined policy might say that instead of outright blocking a behavior, the user might be asked for confirmation. In such case the user would be presented with a human-readable description of the risks, in context relevant to her.

While the concept is general, in security domain we want to carefully choose the scope and ensure the implementation is resistant to tampering. Because overall system security is as weak as its weakest link, the mined rules should cover the secured program interfaces thoroughly. It does no good to block only a selected subset of OS API: All security-relevant API calls have to be protected, and no alternative way of calling them should be left uncovered by the sandbox. Similarly, the inputs should be generated on a system-level, not unit level. Otherwise it will be next to impossible to comprehensively cover common, necessary behavior of the program being secured. Further discussion of constraints, limitations and offsets of sandbox mining is given in next sections in this chapter.

For proof-of-concept implementation and evaluation of sandbox mining we have chosen Android platform. It is popular, its sources are available, it has modern implementation and enables easy access to thousands of real-world apps. We explore Android apps by automatically clicking on their GUI. We monitor how the apps interact with the Android framework API after given GUI actions happened. We codify these interactions in sandbox policies. The GUI input generation is done with our test generator, DROIDMATE. It is described in Chapter 4. The policy inference from mined behavior is implemented via BOXMATE, as discussed in Chapter 5.

3.2 Principles

The sandbox mining concept gives rise to two powerful principles: The *test complement exclusion* principle and the *disclose or die* dilemma.

3.2.1 Test complement exclusion

During sandbox mining phase the mined program behavior is recorded. This is done by running the program with automatically generated inputs, possibly codified in tests. This leads to observable executions of the mined program. From monitoring these executions we obtain an exploration log, which contains mined program behavior. The mined behavior is then codified in a policy which is then enforced at runtime by a sandbox. As a consequence, anything which is not permitted by the policy ultimately derived from executing the tests, or, in other words, the test *complement*, is *excluded from happening without flagging*. By *flagging* an excluded behavior we ensure we can check the behavior is *expected* just before it happens and if not, *exclude* it from happening. We call this trait of sandbox mining idea the *test complement exclusion* principle.

This principle stands in stark contrast to existing automated security approaches, as described in Chapter 2. All of the existing automated approaches are *black-listing malicious behavior*, while our principle is *white-listing mined behavior*. Sandbox mining thus *prevents unexpected behavior change*. If an application would want to behave differently during production than in mining phase, the new behavior would not have been white-listed during testing and would be flagged for review for the user to determine if the behavior is expected or if instead it should be blocked.

3.2.2 Disclose or die

The benefits of test complement exclusion are far-reaching. While with traditional approaches attacker just needs to avoid being black-listed, in case of sandbox mining attacker has to ensure the desired malicious behavior gets white-listed; otherwise it won't ever execute covertly. This is a considerable obstacle for mounting an attack, because the core tenet of malware is that its behavior is *hidden from the user*. But if the attack is to be white-listed, it has to happen already during the mining phase, where it will immediately become visible for automatic or manual assessment, before it can do any actual harm.

Ultimately, the malware writer is put into a dilemma we call the *disclose or die* principle: Either the malware will disclose during mining its intended malicious behavior, where it can be inspected and can do no harm, or it will undergo additional highly contextualized scrutiny at runtime, as anything not seen during mining would be a behavior change, which is reviewed and likely prevented, as highlighted by the test complement exclusion principle.

3.3 Chain of trust

Sandbox mining is heavily inspired by the existing approaches, as listed in Chapter 2. Comparison to them is in order, especially that there are many natural synergies. While sandbox mining can stand on its own, the intermediate artifact it generates, the *mined behavior specification*, can amplify known methods to great effect and possibly enable entirely new security schemes.

To compare our technique to the variety of existing methods, let us setup a concrete scenario serving as a basis of further discussion. Consider *a program developer* that submits her app to the app store. In the app store the app undergoes an audit by *the auditor* to adhere to the store's strict rules, hopefully ensuring the app is secure. If it passes the audit it becomes certified and available to *the end-user*. While running on the end-user phone, the app is encased in a mandatory sandbox with minimal security rules, but additional safeguards can be put in place.

Every good actor in such setup has stakes in ensuring the app is secure. If the developer doesn't ensure it, the app will violate one of the store rules and will be rejected by the auditor. If the auditor certifies malicious software, the app store may incur costs due to user harm. Users do not want to be harmed by improperly certified malware due to reasons listed in Section 2.1.

Unfortunately, today the task of none of these actors is easy. It is hard to construct secure software, prove it is secure, and protect against malicious software, as explained in Chapter 2. How sandbox mining can help here? The three aforementioned actors form a *chain of trust*. The more actors we can trust in the chain of trust and the more supporting security methods we can employ, the more beneficial sandbox mining will be.

3.3.1 End-user

Let's start with the simplest scenario: The end-user puts very limited trust in anybody, except for her sandbox mining solution and OS security rules enforcement implementation, the only security methods she is willing to leverage due to their easy setup and convenience of use. Even in such restricted scenario sandbox mining can prove valuable. Given a potentially malicious, untrusted app, the user can mine the app for behavior enforcement policy. The user can then review the policy for any suspicious behaviors and exclude it. Furthermore, if the user excluded too much, she will have a chance to loosen up the policy during runtime when the behavior is flagged for review.

As a direct consequence of the test complement exclusion principle, behavior changes won't go unnoticed, thus any vulnerability exploitations, malware infections, targeted attacks or plain software bugs won't suddenly wreak havoc upon unsuspecting user. A special case of unexpected behavior change is functionality hidden by design, like backdoors, advanced persistent threats, time bombs or other latent malware. All of these cases will have to withstand human scrutiny before activating, as pointed out by the disclose or die dilemma.

3.3.2 Checksums and certificates

The end-user-only setup can be advantageous against existing methods operating under similar assumptions: Checksums, certificates and user-controlled policies.

Checksums are a "yes/no" proposition: If the user trusts her checksum integrity check and the 3rd party providing the checksum she checks against, she doesn't need sandbox mining. However, this is a very strong assumption, in practice akin to agreeing to terms of use without reading them, only to be surprised after the fact that the user's private data has been shared with the 3rd party. Certificates are very similar to checksums in the sense of being a "trusted/untrusted" determination. They aim to provide more trusted vetting of the app provider, but in our current setup of limited trust there is not much difference. User-controller policies are step-up over a binary choice: The operating system can ask the user which specific permissions to give to the app. However, such policies are not custom-tailored to apps and usually end up being too broad, as discussed in Subsection 2.3.1.

The advantage of sandbox mining over the aforementioned methods is that it provides a *detailed behavior specification of an application, tailored specifically to it.* It is no longer a binary choice - now the user can see what exact behaviors the app wants to be able to do. The specification is mined from specific application and black-lists (flags) unseen behaviors, providing significantly stricter security than broad-stroke facilities of generic user-controlled policies. Even if the specification will turn out to be too strict, the user can relax it when an expected behavior is flagged. This would not be possible e.g. with the usual user/admin split in operating systems: If given operation is not possible as an user, either it is blocked altogether or the app has to gain administrative rights, throwing security out of the window.

All of the sandbox mining benefits are possible on existing software without significant setup effort. Once the end-user does the one-time sandbox mining

infrastructure configuration, mining, reviewing and applying behavior enforcement policies is straightforward.

When the user is asked to endorse a flagged rule violation, she will have appropriate context. As an example, if the secured application will suddenly try to send a text message to a premium number, the user will know she didn't try to send any message, and thus will consider the request highly suspicious. Still, the main challenge of this limited-trust scenario is the difficulty of comprehending the mined behavior specification by the end-user. The specification might prove too hard to reason about, leading her to making it too loose and erroneously allowing malicious behaviors. Similarly, any flagged, actually malicious behavior might end up being allowed by the user, without her being able to appropriately determine its harmful nature. We elaborate on these and relevant challenges in Section 3.5.

3.3.3 Certification levels of trust

Let us now assume the end-user trusts in certificates at least to some degree. Now the burden of securing the application can be shared between the end-user and the auditor providing the certificate. Auditor can *mine* the app and publish it to the app store together with the *app behavior enforcement policy*, for the end-user to *sandbox* during normal use. End-user can then review at runtime any flagged policy violations.

This setup mitigates the issue of end-user having to be proficient with reading the mined specification. The user can behave differently depending on the level of trust given to the certified behavior enforcement policy.

If the trust is complete, the user doesn't question the policy, just applies it. In addition, any flagged behavior is handled completely automatically and the user has no say in it. Depending on the certified enforcement policy provided by the auditor, flagged behavior might be either outright blocked or allowed, but with telemetry information sent to the auditor.

If the behavior is blocked and if it makes the app less usable without any perceived risk, the user might request for the policy provider to relax the policy. The flagged behavior can be automatically allowed if it is suspicious, but unlikely to cause significant harm, and likely to make the app significantly less usable. In other words, when the auditor was unable to make clear-cut decision if to allow it due to involved trade-offs, and decided to gather additional information from production usage. This of course exposes user to some risk, similar to the "catch-22" problem described in Subsection 2.5.3. This time, however, we hope to be able to limit such risks to a small number of corner cases.

If the end-user puts less trust in the certified policy she might take more responsibility over handling the flagged behaviors: Allowing situations that seem harmless to her, and blocking otherwise. Note that because now we assume we operate within the context of a vetted policy provided by expert auditor, it is much more likely any flagged behavior will be easier to reason about and increase the chance user will make correct decision. In any case, the more precise and readable the underlying behavior specification the higher the security and lower app usability loss, as discussed in Section 3.5.

Assuming even less trust brings us into the territory of the user reviewing the policy itself, not only the flagged behavior. However, again, she is better off than if she would have to mine the policy herself. The user can review the certified policy as-is. However, she can also do differential analysis between the certified policy provided by the auditor and by a sandbox mined by herself, focusing on any reported behavior differences.

Note that application updates often require more permissions, as the programs gain more features. With sandbox mining this doesn't mean starting entire security audit from scratch - the end-user can compare behavior enforcement policies of new and old version of the same app - be it certified policies or mined by herself. Such differential analysis will make it blatant what new behaviors the application will start to exhibit after the update.

As we see from the above discussion it is not clear-cut who is responsible for what and many responsibility schemes are possible. By shifting more responsibility to the auditor, we avoid the issue of end-user being unable to accurately handle the flagged behaviors, but require from the auditor to decide how to handle any policy violations ahead of time, without the precise runtime context available to the end-user, as discussed in Subsection 3.3.4. So far we focused on discussing the user side of the story, but let's not forget our goal with sandbox mining is to minimize manual effort, including the manual effort of the auditor providing the certified policy.

3.3.4 Auditor

For auditor the biggest advantage of sandbox mining is being able to audit a mined app behavior specification instead of the app's source code. Reviewing just the mined behavior specification is enough, as postulated by the test complement exclusion principle: Any behavior not present in the specification will be flagged in production.

One can argue we end up in the same situation as end-user: given an application, the auditor has to determine its harmlessness. However, we can make

a set of important assumptions about the auditor that we couldn't make about the user. The auditor is an expert in assessing and adjusting the mined specifications; she can incentivize application developers to make her work easier as discussed in Subsection 3.3.5; has the expertise to leverage any other security methods as explained in Section 3.4; she has access to database of other applications and their mined behavior specifications as elaborated on in Section 3.6; and her task is to conduct the audits, while for the end-user this was just an additional effort detracting from the experience.

If the auditor would want to use only sandbox mining, her procedure would be similar to that of end-user described in Subsection 3.3.1 with the important difference of reacting to flagged behavior. The auditor has an additional responsibility of determining how to automatically handle behaviors violating the policy on behalf of the end-user. In simplest scenario all of them could be blocked, but that might decrease application usability. Allowing them silently doesn't make sense: If they are to be allowed silently, they should have been included in the policy in the first place, thus their absence denotes the policy deficiency. In such cases appropriate telemetry should be gathered to reclassify such alerts in the future. The remaining gray area are those flagged behaviors that are left for the user to decide because they are too hard to decide on by auditor ahead of time. Given such scenario, the auditor should ensure that when the rule is violated, the user is provided with as much as possible relevant, readable information, to help her make the right decision: Block harmful behavior, while avoiding blocking expected behavior, thus avoiding decreasing app usability.

In the chain of trust schema it is only reasonable we cannot trust the developer of the application or the developer is unknown altogether. However, it is possible to make auditor life easier by appropriate developer incentives, as described in Subsection 3.3.5.

3.3.5 Developer

A white-hat developer can benefit from our technique. Recall that the developed app needs to pass the security audit done by auditor to be admitted to the app store. The admittance process to existing app stores like Apple App Store or Google Play Store leave a lot to be desired. Apple App Store has stringent set of rules and apps often get rejected without adequate explanation and after long delays. Apple prefers to err on the side of user security and employs human auditors, delaying the process. Google Play Store, on the other hand, has automated and secret methods to analyze apps, which leads to occasional

unexpected rejections, but mostly many malicious apps passing through.

Sandbox mining would enable app store to audit based on the mined specifications instead of the raw app artifacts. This enables app store to incentivize developers to provide their own mined specifications alongside the apps themselves. Developers can submit apps for review together with the mined behavior, annotated with their remarks explaining which parts of the mined behavior are required for which feature. This way the developer can ensure all the required behavior will be allowed and will make the auditor's job easier by providing additional context.

Developers that consistently submit automatically mined well-documented high-quality behavior specifications for their new apps and app updates can significantly cut down on the time auditor needs to spend reviewing their apps. As a result, such developers could gain reputation with the app store and have their reviews expedited, to incentivize them to provide high quality specs.

Of course a malicious developer could try to game the system by first gaining trust of the auditor. The developer could leverage the trust to justify approving a behavior that is actually malicious. This, however, would require significant effort from the malware writer. Furthermore, the auditor should always do her due diligence by re-mining the application sandbox on her own machine and comparing it with the one provided by the developer. We do not expect for the sandboxes to be the same due to the issues described in Section 3.5, but still, by doing differential analysis of the mined behaviors any potentially malicious discrepancies have a higher chance to stand out. Such discrepancies could then be reported back to the developer for explanation and adjustment.

3.4 Synergies

While sandbox mining can stand on its own, as discussed in the above-given sections, the technique has potential for powerful synergies with existing automated approaches to securing software.

3.4.1 Dynamic analysis

First off, mining of application behavior requires input generator, which is also a prerequisite to dynamic analysis. Thus, any work done on generating better inputs will benefit both dynamic analysis techniques and sandbox mining. While input generation for dynamic analysis has many downsides as described

in Chapter 2, leveraging it for sandbox mining can significantly mitigate many of them.

In pure input generation, as used for dynamic analysis, its *incompleteness* is a major issue preventing us from having meaningful descriptions of programs. However, for sandbox mining this is not necessarily a bad thing - by excluding most behaviors, we also exclude malicious actions, as codified by the test complement exclusion principle. Of course, the challenge of ensuring desired behavior is not excluded, i.e. the challenge of underapproximation, remains. For that we employ our chain of trust setup given in Section 3.3. First, the developer can mine the sandbox and review it for any omissions. Next, the auditor can review the mined behavior harmlessness. Finally, if something was still missed, the end-user may either add any flagged behaviors by herself, or request addition to the auditor or developer.

Another issue of standard dynamic analysis are time bombs and other evasion techniques. However, with the disclose or die principle afforded by the chain of trust, the hybrid of input generation and sandbox mining has potential to avoid these issues while significantly neutralizing the usability concern, as described in previous paragraphs.

3.4.2 Static analysis

It is known that leveraging static analysis techniques for input generation can be beneficial, as e.g. it is the case with dynamic symbolic execution described in Section 2.5.2. However, static analysis faces many issues as discussed in Chapter 2, including program complexity, obfuscation, encryption, information erasure in object code, overapproximation and even undecidability. However, because in the context of sandbox mining we would use static analysis only to fine-tune mined behaviors, any countermeasures to static analysis employed by malware writers will have less impact. We discuss possible challenges in detail in Section 3.5.

3.4.3 Runtime enforcement

Sandbox mining is partially a runtime enforcement technique, as the second phase uses a sandbox enforcing the mined behavior policy. Other runtime enforcement techniques include anti-virus programs, firewalls and network monitoring. Firewalls require policies, which sandbox mining provides. Anti-virus programs require malware signatures, crippling their effectiveness against novel malware. Sandbox mining is not hindered by new malware and doesn't require

any signatures. However, signatures could be used to improve the mined polices, as discussed in Section 3.6.

Sandbox mining also doesn't have to do precise, potentially privacy-breaching network packet inspection, as it can be used in isolation the end-user machine, not a possibly eavesdropping middle man. However, gathering telemetry on gray area corner cases as discussed in Subsection 3.3.4 could help refine the sandboxes, at cost of ensuring that end-user privacy is not breached.

3.4.4 Anomaly detection

Anomaly detection systems have to be trained in production to build a model of allowed behavior, putting users at risk while the model is not yet available, violating their privacy and potentially degrading user experience due to performance overhead. These problems are significantly mitigated by the chain of trust, as introduced in Section 3.3. That is, a hybrid of automatic mining of behavior policies coupled with additional manual scrutiny of the ambiguous cases ensures the attack surface on the user is minimal, instead of leaving the users unprotected until the model is available.

3.5 Mined behavior specificity trade-off

The single most important characteristic of sandbox mining, what makes it or breaks it, is the quality of the mined behavior specification and derived enforcement policy. Ideally we would like to have a specification that is very comprehensive, including all the required behaviors. At the same time, the specification should be very precise, i.e. it should ensure than no strictly necessary behaviors are allowed, analogously to the principle of least privilege [55]. On top of that, the specification should be easy to read so humans can easily review it, extend, compare or modify. Finally, all of that has to be achievable on existing infrastructure, with acceptable (that is, negligible) performance impact and require reasonable effort to implement.

This thesis tries to answer how close we are to that ideal by posing research questions in Section 3.7. For discussion in this section, let us assume we can obtain comprehensive specifications with reasonable effort. This will allow us to elaborate on the main trade-off of mined specifications: *Precision vs readability.*

The existing user-controlled policies, as described in Subsection 2.3.1, can be considered a very crude form of behavior specifications. For example, the specification can say "this application requires administrative privileges" or "this

application requires access to your camera". Such statements are easy enough to understand we expect a significant amount of end-users to be able to effectively work with them. For example, consider a behavior enforcement policy of a note-taking application that doesn't include access to microphone. Any attempt to enable microphone recording would be flagged and result in a request being made to the user to allow the app access to the microphone. We can expect some users will correctly decline it, unless, of course, the application has voice note-taking function that the user wanted to use.

Unfortunately, such coarse-grained behaviors leave huge attack surface for the attacker. More advanced applications might require access to specific sensitive resources only for one component and for very specific parameters. For example, a game may require access to send premium SMS message to specific number only when the end-user clicks a button to buy a power-up. However, if all you can do is to allow sending all SMS messages to all numbers or block it completely, you do not allow the user any choice that keeps her safe and at the same enables her to buy the power-ups.

What you need is more *specific* behavior description. For example, you could *contextualize* given resource access to GUI buttons. This way the user could allow to send premium SMS messages only if the "confirm transaction" button is clicked. Such level of precision indeed seems to strike good balance between specificity, reducing attack surface, and readability. We describe such specifications in a systematic way in Section 5.3.

There are many more levels of behavior scope fine-graininess, based in a large degree on the domain we operate in, with examples given in Subsection 3.1.2. One could even image restricting access to specific OS Kernel calls from specific high-level methods, but such specifications would be unreadable.

It would be excellent to be able to flag a behavior with information like "Once you click *confirm transaction* after clicking *buy power-up* the application will send SMS message to number 123-456-7890, certified to belong to the application power-up center. This will cost you 7 USD. Approve?". However, automatically gathering information and presenting it in human-readable form is challenging. If we monitor application API we might observe a method call like `Sms.Send()`. The method call sends an SMS message to a specific number according to the state mutations done on `Sms` object by previous method calls. Automatically extracting the SMS information to present it to the user in human-readable form might thus require help of static analysis, as discussed in Subsection 3.4.2. Such analysis, in turn, can be prevented by usual static analysis countermeasures.

Reliably identifying given button is also nontrivial. The *confirm transaction* button mentioned in previous paragraph might not be unique: Multiple buttons

might have the same label, or it might even just say *confirm*. Often we do not have access to the button GUID and thus have to rely on surrounding GUI screen structure to disambiguate between different buttons. The GUI screen structure might contain dynamic elements, like list of recent messages, making it hard to determine if we are on the same or different screen, and thus if we are dealing with the same or different buttons.

Because in the given example we are relying on fairly low-level of abstraction of API method calls and their inputs, the mining process can be sensitive to the environment in which it runs. As a result, the same app might behave differently on different computers due to different file system structure, random seeds, network availability, performance characteristics, time of day, etc. Even two consecutive executions of mining phase in the same environment might lead to different results, and we need to be able to abstract over the noisy differences.

All these challenges might lead us to make simplifications, e.g. treating all buttons with *confirm* label to be the same. This, however, opens an attack opportunity to the attacker. For example, once the user *confirms* she wants to share her high score via SMS with her friend, the behavior policy might assume she *confirms* sending any SMS, including sending premium messages to malicious number.

In extreme case one could even imagine application with heavily obfuscated code, defeating static analysis, that also obfuscates its entire interaction with environment: All files written to disk have nonsensical names in nonsensical directories, all network messages content is scrambled and/or encrypted, etc. Any mined behaviors in such cases will be utterly unreadable and could not be easily reverse-engineered due to the obfuscation. On the plus side, all of this should raise red flags to the auditor, thus underlying importance of disclose or die principle: This time however no attack was disclosed, but *a clear intent to hide one*.

Overall, fine-tuning the mined behavior specificity against its readability is an important trade-off right at the center of sandbox mining that might be used to defeat the method. However, let us emphasize this is a significant improvement over the state-of-the-art of binary checksums, certificates, user/admin permissions, user-based policies and other methods previously discussed.

3.6 Marketplaces

Preceding sections discussed in detail how sandbox mining fits in the landscape of existing security tools. The rest of this thesis focuses on proving principal

feasibility of the idea. In this section we focus on exploring what wide adoption of sandbox mining may bring.

In the chain of trust we proposed we introduced three actors: The developer, the auditor and the end-user. However, it helps to think about them in terms of *roles*, representing global groups of people. The existing app stores are already benefiting from the economies of scale, where patterns common among thousands of apps can be compared to detect any anomalies, like e.g. shown with the CHABADA work [31]. One can easily imagine the same can be applied to the mined program behavior specifications. Comparing any new mined behavior against the database of known thousands of specifications and their fragments can expedite the assessment of the new behavior by highlighting anomalies. Possibly the mined behavior is composed of a sum of known OS access patterns, commonly used by other, approved application behaviors and well-understood. For example, applications might want to keep their transient cache in `<user_home>/<app_name>/cache` directory. This cross-app comparison for good and bad behaviors fulfills similar role to signatures of known viruses for anti-virus programs.

In case of behaviors that are hard to classify ahead of time by auditor, additional telemetry from end-user usage and policy violations can be gathered to review the problematic behaviors in context of real-world usage and make a concrete decision to block, allow or perhaps split into two or more sub-rules with different handling. All of this can feed to global knowledge base on application behavior specifications.

Cross-app mined behavior specification comparisons are feasible only if majority of applications are published together with their mined specifications. So far this was infeasible to assume due to enormous manual effort of precisely specifying what an application can do. However, with sandbox mining providing the baseline spec that then has to be post-processed by the developer and/or auditor, this is brought in the realm of possibility.

As the idea of applications always having their specifications available catches on, programs without them can be penalized or outright rejected from admission to app stores. This might lead us to assume the specification is always available, and its lack is a major red flag. Ubiquitously available specifications could then be shared with end-users via dedicated *behavior enforcement policy marketplaces*. Such marketplaces would be similar to certification authorities, but the users would not have to blindly trust in the certificate value, but actually review the behavior enforcement policy. If the end-user doesn't trust one certificate authority and doesn't have the expertise to review the policy, she could hire an independent auditor to review the policy for her. It is possible one

app could have multiple enforcement policies with varying level of strictness, allowing more privacy and security-conscious users to use the more stringent policies.

The assumption that any application has to come with semi-automatically mined specification could open-up global thinking to sharing applications with additional metadata. Currently most program are provided as-is, possibly with a checksum, a certificate, or, in the rare case the application is open-source, a link to the repository with source code plus possibly a unit test suite and build instructions. However, once we will expect from all the applications to always publish a specification, why not also expect to provide the set of generated inputs used to generate these specifications and behavior enforcement policies?

Sharing generated inputs globally can multiply the efficiency of sandbox mining. Instead of everyone having to rely on input generator to provide the inputs from scratch, one could reuse existing, certified inputs that efficiently test an application in various scenarios. If we are providing the inputs, we could also provide the environment details, to allow as precise replication of mined policy as possible, similarly to the idea of reproducible builds.

As more advanced techniques are developed, more specialized application metadata can be conceived and we might build up infrastructure for sharing the programs with arbitrary number of specialized, derived artifacts.

Overall, sandbox mining, if adopted, has a potential to make everybody considerably more secure by shifting our expectations from applications we use from being black-boxes to have their specifications always included and possibly more, while not incurring any significant additional costs.

3.7 Challenges and research questions

As introduced in Section 3.5, sandbox mining feasibility hinges on the quality and applicability of the mined behavior specifications. We can say that for the sandbox mining to work, four important assumptions have to hold. Truthfulness of each of these assumptions might challenge the viability of the method, and thus each of the assumptions warrants a research question that needs to be answered. In the context of chain of trust introduced in Section 3.3, unless noted otherwise, we will be posing the research questions in the simplest scenario in which the *end-user* has to conduct entire sandbox mining process herself, without the help of any other methods or audited/certified behavior enforcement policies, as detailed in Subsection 3.3.1.

First, the employed test generator has to be powerful enough to cover all

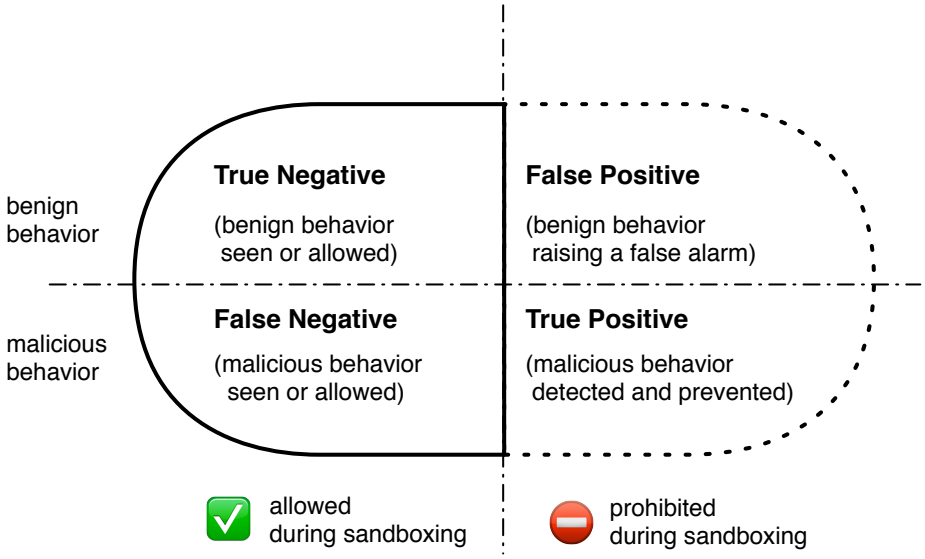

Figure 3.2: Confusion matrix. Program behavior is either benign or malicious; if it is not seen during mining phase, it is prohibited during sandboxing. The three risks are *false positives* (benign behavior not seen during mining and thus requiring confirmation during sandboxing), *false negatives allowed* (malicious behavior allowed because of too coarse sandbox rules), and *false negatives seen* (malicious behavior seen during mining, but not recognized as such, and thus allowed).

the *common, benign behavior* exhibited by the application being secured. If this would not be the case, the behavior policy would not cover some commonly used benign behaviors. As a consequence, they would be outright blocked or would at least require user intervention. This would make the program potentially unusable, thus failing our requirement of the approach being *useful in practice*, as outlined in Section 2.2. In other words, we would end up with too many *false positives*, as explained on Figure 3.2. To confirm we have good enough test generators, we pose our first question:

Q1 *Can input generators sufficiently cover behavior?*

Second, the mined behavior has to be fine-grained enough, otherwise the attack surface will remain too big. Recall that manually provided behavior policies have to be coarse-grained, otherwise it would be too impractical to

define them. As a simple example, consider Android permission groups [83]. To reiterate arguments made in Section 3.5: As soon as the user allows given application to read SMS messages, she allows the app to also send any number of SMS messages, to any address, at any time, also covertly in the background. Attacker could easily exploit it to steal information or deplete account of the user by dialing premium numbers. Mining sandboxes promise is to provide more fine-grained behavior distinction, thus reducing the attack surface, resulting in smaller number of *false negatives allowed*, as visualized in Figure 3.2. We can check the quality of mined behaviors by answering the following question:

Q2 *Can we reduce the attack surface by providing a behavior enforcement policy more fine-grained than Android permission system?*

Third, as we obtain fine-grained behavior, we run into the risk of it being incomprehensible to the user. Recall that if a sandbox-violating behavior occurs, it might be blocked or presented to the end-user for endorsement. If the suspicious behavior description is unclear to the user, she might end misclassifying it: She might forbid benign behavior, decreasing app usability and resulting in a *false positive*, or she might allow malicious behavior, putting herself in harm's way and resulting in a *false negative*.

There is also another risk here: As the behavior gets mined for the first time, it might actually allow malicious behavior. It might happen because the malware writer, being forced into the disclose or die dilemma, might try to hide her intentions in plain sight by having them added to the allowed behaviors.

Q3 *Can the more fine-grained mined behavior help users and experts correctly classify behavior as benign or malicious?*

Finally, in Section 2.2 we also stated our solution has to be *widely applicable* on *existing software*. In our context this means the tools we use should be able to fully automatically mine sandboxes from a variety of most popular, most used Android apps. This might be challenging, as such apps are usually complex and full automation of sandbox mining implies ability to automatically handle many corner cases, like the app being secured crashing at unexpected times. To determine if we possess good enough tools, we will answer our final question:

Q4 *Can modern input generators be successfully and fully automatically used to mine sandboxes from a variety of existing, widely used applications?*

If we answer the four stated questions positively, we will prove sandbox mining meets the assumptions required to make the approach work. We will also

prove sandbox mining fulfills all the criteria described in Section 2.2. Indeed, our implementation and its evaluation, as described in next chapters, is easy to setup and works with existing Android apps, thus ensuring the approach is *useful in practice* and *widely applicable*.

Chapter 4

Input generation implementation: DroidMate

To answers research questions posed in Section 3.7 we need a concrete implementation of the sandbox mining concept. Three components are required: An *automated input generator*, a *sandbox policy inference engine* and a *sandbox* that will enforce the inferred policy at runtime.

DROIDMATE, described in this section, is our implementation of the automated input generator for Android 4.4 (API 19) and 6 (API 23). It is a GUI interaction input generator employing biased random *exploration strategy* aware of the GUI structure. The *exploration strategy* is the algorithm deciding which inputs to generate. The remaining two components are implemented by our sandboxing solution, BOXMATE, described in Chapter 5. While conceptually separate, both of these tools share common codebase.

4.1 Technical choices justification

Why Android? We have chosen our implementation to be on Android for several reasons. Android is the most popular mobile OS in the world with thousands of widely used apps, giving us plenty of opportunity to evaluate sandbox mining viability, answer our research questions and conduct future research. All these applications are easily obtainable from Google Play Store [104]. Furthermore, vast majority of them have nontrivial GUI, meaning the input generator we write can be a GUI interaction input generator. Android is also an open-

source software, making building any tools based on it significantly easier. This has also resulted in it being a very fertile ground for various research tools and projects, as discussed in Section 4.2. We can reuse knowledge codified in their implementation and published research results for our own goals.

Why GUI input generator? Using GUI as the exercised system-level interface to the program being secured has two important benefits. First, relatively short sequences of GUI interactions can trigger large swaths of realistic program behaviors. Second, any sandbox policies formed pertaining to GUI have big chance to be intuitive to human users, in case a decision has to be made to endorse a behavior not seen during mining.

Why biased random exploration strategy aware of GUI structure? Such strategy provides, at only modest implementation effort, significant improvement in efficiency at discovering diverse behaviors of application being secured, as compared to a completely random strategy. As a result, comprehensive behaviors are mined much faster. The GUI structure awareness allows us to focus only on GUI elements which may result in interesting behavior when interacted with. For example, we will click only on elements that have their *clickable* XML attribute set to true. The fact exploration is biased ensures discovering new behavior takes precedence over repeatedly covering the behavior already known.

Why not reuse existing test generator? There are already many Android GUI-based test generators in existence. However, no existing GUI test generator fulfills all of our requirements, which are:

Built-in support for monitoring Most importantly, we require from our input generator to have a built-in support for a detailed, low-overhead monitoring of Android framework API method calls, which is required to construct BOXMATE-enforced policies, as described in Chapter 5. None of the existing tools have that capability.

Efficient The used GUI test generator should explore the GUI quickly, so mining sandbox rules doesn't take too much time overall. This entails the generator has to explore the GUI at least semi-systematically, not purely randomly.

Robust The tool should be robust, being able to work even with the most complex apps on the market. This way we can conduct experiments with

many representative, relevant samples, reducing threats to external validity. In addition, the generator should be able to provide inputs for many apps in one continuous session, even while unattended. If some of the apps crash it should recover and continue. This way it can be setup once for overnight runs on many applications to gather experiment data for the next day, week or even month.

Sources not needed As the tool is expected to work on any downloaded application, we cannot assume we have access to the source code. Thus, the input generator has to work on the raw application file (its .apk file), without access to any sources.

Relevant We want for the generator to work on newest Android OS. Every year new major Android OS version is released, sometimes with significant changes to the permission model, having major impact on the mined rules. Not having a test generator up-to-date with current Android version puts any results obtained at risk of rapidly becoming obsolete. In the time frame this research was conducted the newest version of Android was 6 (API 23).

Easy to setup We want for the test generator to be easily deployable, to show end-users can use it relatively easily. In particular, the tool has to work without any modifications to the Android OS, without the need to install any 3rd party frameworks and even without rooting the device.

Easy to develop and extend Finally, we want for our tool to serve as a development platform for further research and extensions. To this end, we made it extensible, with thorough automated test suite, documented, automatically integrated and open-source [102].

Section 4.2 surveys existing Android GUI test generators and explains which of the above given criteria they do not fulfill.

4.2 Comparison with existing tools

In this section we survey state-of-the-art of existing GUI test generators for Android. We highlight some of the tools, comparing them in detail to our requirements as described in previous Section 4.1.

We list the tools chronologically by their publication date to highlight how the field evolved over time. Note that none of the tools has a built-in support

for monitoring Android framework API method calls to the level of detail and extensive coverage of API surface as provided by DROIDMATE, while maintaining the app stability. This is a critical requirement for us. As all the tools listed here are undocumented and mostly unmaintained research prototypes, extending any of them to add such capability would require prohibitive effort. Moreover, sources of some of the tools are not publicly available.

Android Monkey [100] is a GUI stress-testing tool which comes built into the Android SDK, which was first released in October 2009 [98]. Its main disadvantage is it doesn't explore the GUI systematically. It is aware only of the screen size and inputs the GUI events at random coordinates, checking if the application didn't crash.

SwiftHand by Choi et al. [18, 106], published May 2011, is an early input generator that uses machine learning to learn a model of the app during testing, uses the learned model to generate user inputs that visit unexplored states of the app, and uses the execution of the app on the generated inputs to refine the model. Unfortunately, the most recent Android version it supports is only 4.1.

ACTEve by Anand et al. [5, 108], published November 2012, uses dynamic symbolic execution (DSE) to find sequences of GUI tap events which maximize the coverage of the app's bytecode. ACTEve alleviates the path explosion problem of DSE by identifying subsumption between different event sequences and pruning the redundant sequences. Unfortunately, ACTEve requires instrumentation of the Android SDK, supports only GUI taps and has been evaluated only on a small set of 5 apps, hinting the proof-of-concept implementation would require significant work to meet our robustness goal. Given the tool hasn't been updated for over 4 years, most likely it would require additional significant effort to adapt to current Android version.

Dynodroid by MacHiry et al. [46, 107], published August 2013, explores the GUI semi-systematically. Not only it recognizes the GUI structure and is aware which elements are actually clickable, it also biases its random clicking strategy to prioritize interacting with new elements. This way it explores new behaviors faster. DROIDMATE exploration strategy, as described in Subsection 4.7.1, is based on Dynodroid. In addition to GUI events, Dynodroid can generate *system events*, like incoming SMS message. Another interesting feature is that the input generation can be interleaved with manually provided inputs: User can at any point stop the generation, guide the exploration by doing manual actions, and resume automatic generation.

Dynodroid has two major flaws that disqualify its usage for our purposes: First, it requires modification of the underlying OS. This is very cumbersome and time consuming task requiring a lot of expertise, thus it breaks our require-

ment of being easy to setup. Secondly, Dynodroid works only on the severely outdated Android 2.3.5 and is no longer maintained, thus making any results obtained with it obsolete, not fulfilling another of our needs.

A³E by Azim and Neamtiu [8, 105], published October 2013, uses static, taint-style, dataflow analysis on the app bytecode to construct a high-level control flow graph that captures legal transitions among activities (app screens). One exploration strategy employed by A³E is clicking on the GUI systematically, focusing on covering the static graph, to quickly and systematically cover as much screens as possible. Second strategy focuses on prioritizing launching screens present in the static graph which would be hard or even impossible to test by plain user interaction with the app's GUI, because they are, for example, triggered by other applications or background services. This is done by constructing and sending appropriate system event (so called *intent*). We decided against using A³E as it is untested on Android ≥ 5 and has no support for monitoring Android framework API method calls.

PUMA by Hao et al. [34, 74], published June 2014, is a flexible framework for implementing various state-based test strategies. Even though PUMA was evaluated on 3,600 apps, hinting at good scalability, it supports only 9,644 out of 18,962 checked apps (51%), due to limitations of bytecode conversion and unsupported app categories like games. This violates our requirement of being able to support more complex apps, for which usually the bytecode conversion fails. In addition, PUMA is geared towards operators of app marketplaces, not end-users.

ANDLANDIS by Bierma et al. [13], published October 2014, is a scalable dynamic analysis system capable of processing over 3000 Android applications per hour. ANDLANDIS feeds simulated data to sensors (GPS, camera, motion, touch, screen, etc.) and intercepts outgoing traffic (Internet, SMS, phone, etc.) for forensic analysis. There are not many details given on the exploration strategy, except that it attempts to visit as many features of the application as possible by prioritizing GUI elements which were not yet interacted with. While ANDLANDIS scalability actually goes far beyond our requirements, it does so at a cost of complex setup: The tool is designed to work on a parallelizable distributed machine cluster, while we aim to provide a solution that can be easily setup by an end-user on his personal computer and device.

SAPIENZ by Mao et al. [48], published July 2016, is a modern test generator that uses multi-objective search-based exploration strategy. SAPIENZ combines random fuzzing, systematic and search-based exploration, exploiting seeding and multi-level instrumentation. The first reason we didn't use SAPIENZ is that it was developed too late for the time frame in which research for this thesis

was conducted. Even if that would not be the case, SAPIENZ still suffers from many of the same problems as the other tools: No support for API method calls monitoring and lack of support for Android 6.

CuriousDroid by Carter et al. [17], published May 2017, is similar in many regards to DROIDMATE. It analyzes GUI structure like DROIDMATE and prioritizes randomly exploring yet unknown behaviors. There are differences in the employed algorithms, e.g. CuriousDroid uses heuristics to click widgets on given GUI screen in specific order, e.g. clicking "OK" button last. CuriousDroid modifies the bytecode of the AUE, but using different instrumentation framework. It requires for the device to be rooted, while DROIDMATE doesn't. CuriousDroid sources are only partially available [110], it is unclear how resilient it is against app and device crashes, and like SAPIENZ, was published long after the research for this thesis was conducted.

DroidBot by Li et al. [43, 111], published May 2017, is another generator similar to DROIDMATE. Unlike DROIDMATE it requires no app instrumentation, but doesn't provide such advanced Android framework API method calls monitoring as DROIDMATE.

There are many other automated Android GUI input generators, like AndroidRipper (subsequently MobiGUITAR [3]) by Amalfitano et al. [2], AppsPlayground by Rastogi et al. [53], BRAHMASTRA by Bhoraskar et al. [12], CrashScope by Moran et al. [50], EvoDroid by Mahmood et al. [47], JPF-ANDROID by van der Merwe et al. [59], MonkeyLab by Linares-Vásquez et al. [44], ORBIT by Yang et al. [61], Thor by Adamsen et al. [1] or IntelliDroid by Wong and Lie [60, 109].

A 2015 survey by Choudhary et al. [19, 71] and the related work section of the SAPIENZ paper [48] provide good overview and comparison of these input generators. The most prevalent problems with these tools, in addition to lack of means of monitoring API method calls, are that they: Support only a small set of apps; work only with simple apps; require OS modification, making setup complex; or do not support Android 6 or higher, making results obtained with them outdated, as Android 6 introduced significant changes to its API and permission model [62]. See Subsection 4.2.1 for more systematic comparison.

4.2.1 Comparison table

The Table 4.1 summarizes key differences between DROIDMATE and the tools mentioned in Section 4.2. *Strategy* denotes the kind of exploration the generator is doing on supported *inputs*. The inputs can be GUI, like taps or text input, or (Sys)tem, like Android intents or OS events. The *monitor* column emphasizes

Tool	Strategy	Inputs	Monitor	Apps	Src	OS	Instr.	Open
A^{3}E Depth-first	Model-based	GUI	No	Complex	No	2.3.4	None	Yes
A^{3}E Targeted	Systematic	GUI	No	Complex	No	2.3.4	None	Yes
ACTEve	Systematic	GUI	No	Simple	Yes	2.3	SDK	No
ANDLANDIS	Guided random	GUI	No	Simple	No	4.2 Emu	None	No
AndroidRipper	Model-based	GUI	No	Simple	Yes	4.X	Apps	No
Android Monkey	Random	GUI, Sys	No	Complex	No	Any	None	Yes
AppsPlayground	Guided random	GUI, Sys	No	Complex	No	? Emu	SDK	No
BRAHMASTRA	Model-based	GUI	No	Simple	No	? Emu	Apps	No
CrashScope	Systematic	GUI, Sys	No	Simple	No	?	No	No
CuriousDroid	Guided random	GUI	No	Complex	No	?	Apps	No
DroidBot	Guided random	GUI, Sys	No	Complex	No	Any	No	Yes
DROIDMATE	Guided random	GUI	Yes	Complex	No	4.4.2; 6.X	Apps	CI
Dynodroid	Guided random	GUI, Sys	No	Complex	No	2.3.5	SDK	Yes
EvoDroid	Systematic	GUI	No	Simple	Yes	? Emu	None	No
IntelliDroid	Systematic	Sys	No	Simple	No	4.3	None	Yes
JPF-ANDROID	Systematic	GUI	No	Simple	Yes	?	None	Yes
MobiGUITAR	Model-based	GUI	No	Simple	Yes	4.X	Apps	Yes
MonkeyLab	Model-based	GUI	No	Simple	No	?	No	No
ORBIT	Model-based	GUI	No	Simple	Yes	?	No	No
PUMA	Model-based	GUI, Sys	No	Simple	No	4.1	Apps	Yes
SAPIENZ	Search-based	GUI, Sys	No	Complex	No	4.X	Apps	Yes
SwiftHand	Model-based	GUI	No	Simple	No	4.1	Apps	Yes
TrimDroid	Systematic	GUI	No	Simple	No	?	No	No

Table 4.1: Comparison of DROIDMATE with existing input generators. See Subsection 4.2.1 for legend.

that no other tool can monitor the Android API surface to the extent and with level of detail as required by DROIDMATE, while retaining the app stability due to low intrusiveness of the instrumentation used by DROIDMATE. Based on the published evaluation set and employed instrumentation framework, we classified any given generator as being able to handle either only *simple apps* or *complex apps*. Simple apps are usually manually hand-picked apps for proof-of-concept evaluation. However some tools might be able to handle only simple apps even if the evaluation study was conducted on thousands of apps. This is caused by these tools using instrumentation frameworks that are highly unstable, causing many apps to break. This is the case with e.g. PUMA. *Complex apps* means we found no reason to think given generator won't be able to handle apps on the market that have the most complex bytecode structure. A generator requires

an app *source* ("Src") and supports given Android *OS*. It might work only on *emu*lators. If we were unable to determine, based on the publication or source code (if available), the Android version supported, we denote this with question mark. Some tools might require *instrumentation* either of the explored apps, or the Android framework itself (SDK). Finally, a tool is easy to extend by practitioners if it is *open*-sourced. We denote DROIDMATE as *CI* to signify it is not only open-source, but also has cross-OS Continuous Integration builds and other artifacts available making DROIDMATE considerably easier to run, adopt and extend by other practitioners.

4.3 DroidMate overview

DROIDMATE [103] is an automatic GUI input generator for Android apps. DROIDMATE fully automatically explores behavior of an Android app by interacting with its GUI. DROIDMATE repeatedly reads the device state, makes a decision and interacts with the GUI, until some *termination criterion* is satisfied. While this happens, DROIDMATE *monitors* (records) which Android framework API method calls have been made by the subject app and by which GUI interaction they have been triggered. This process is called an *exploration* of the *application under exploration* (AUE). DROIDMATE is fully automatic: After it has been set up and started, the exploration itself does not require human presence.

DROIDMATE can be run from command line or through its Java API. As input, it reads a directory containing Android apps (in form of .apk files). It outputs a serialized Java object representing all the data obtained from the explorations, that is, it outputs an *exploration log*. It also outputs .txt files and charts having various human-readable information extracted from the exploration log, including its textual summary called *exploration summary*.

DROIDMATE can click and long-click the AUE's GUI, (re)start the AUE, and it can terminate the exploration. Any of this is called an *exploration action*. Details are given in Subsection 4.7.1. DROIDMATE's *exploration strategy* decides which exploration action to execute based on (a) the *XML window hierarchy* of the currently visible device GUI, which it wraps into an object representation called *GUI snapshot*, and (b) the set of Android framework API methods that have been called after last exploration action, i.e. a set of *API method calls*.

DROIDMATE is implemented in JVM languages: Java, Groovy and Kotlin. Most of the code is executed on the *host machine*, i.e. the developer or user machine. Small parts of DROIDMATE are executed on the Android device, as explained in detail in Section 4.8 and Section 4.9. DROIDMATE is built with Gradle

and works on all major operating systems: Windows, macOS and Linux. This thesis describes DROIDMATE as of May 2017, located in GitHub repository [103]. It is composed of over 37,000 nonempty lines of code. This includes comments, sample DROIDMATE API usages, almost 2,000 lines of Gradle build scripts and over 9,000 lines of an automated regression tests. We exclude generated code in the count. Primary development of DROIDMATE was done on Windows 7 and 10. DROIDMATE has continuous integration build done on Linux [103] and has seen multiple successful deployments on macOS.

4.4 Key features

DROIDMATE supports Android 4.4.2 and 6. It is designed to be run overnight on multiple apps without human presence, saving exploration summaries obtained from explorations of all the applications. Thus, DROIDMATE is highly robust, handling many exceptional scenarios caused by the AUE or even by entire Android device crashing. Depending on the concrete type of failure, DROIDMATE will retry last failing operation, restart the AUE, abort exploring the AUE and continue with next one, or even restart the device. When all options are exhausted, DROIDMATE will gracefully terminate the exploration and save any results and crash logs obtained thus far.

DROIDMATE doesn't require Android device to be rooted to work, makes no modification to the underlying OS and makes only minimal modification to the explored apps, called *inlining*, as explained in progressively more detail in Section 4.6, Section 4.9 and Section 4.11. This way even the most complex apps retain their original stability and can be explored by DROIDMATE, as long as they don't do integrity checks, which is very rare.

DROIDMATE provides facilities to determine tested device GUI structure, to act on its GUI elements and to read the details of the Android framework API methods called by the AUE, including their signatures, parameter names and values, and stack traces. All this information becomes immediately available during the exploration, and thus can be used to adapt it on the fly.

Furthermore, by design, the exploration strategy, including its termination criterion, is replaceable without recompiling DROIDMATE itself. Changing the methods being monitored requires trivial .txt file modification and recompilation. Recompilation, after minimal initial setup, is a one step process, detailed in the tool documentation [103]. To sum up: *DROIDMATE is designed to be usable by other researchers and to work reliably even on the most complex of apps.*

4.5 Architecture

Following sections describe the architecture of DROIDMATE, split into following elements: execution phases, components, modules and resources.

Execution phases give a holistic view of how DROIDMATE operates. Starting from reading all its inputs, through overview of the operations DROIDMATE does in sequence, ending at producing all the outputs.

Components are core conceptual entities, spread across many modules and using many resources. We distinguish three components: *exploration component* (Section 4.7), *GUI automation component* (Section 4.8) and *monitoring component* (Section 4.9). Exploration component is the one running the show. The remaining two components exist to support it and can be considered its subcomponents. The host machine part of the exploration component is represented by the `Exploration` class in the *command* module, and all the supporting classes that are being used from within the `Exploration` class. The remainder of the exploration component is present on the device, in form of device parts of the remaining two components.

Both *monitoring* and *GUI automation* components are spread across many modules, with majority of their functionality being located on the Android device, not on the host machine, as opposed to the exploration component. The details are given in Section 4.10.

Modules are compilation units. Each module maps to one file system directory. In Section 4.10 we give overview of each module and their relationships.

Resources are files required at DROIDMATE runtime. The most prominent resources are .apk files of the apps to explore, the same .apk files after inlining, list of monitored API methods and the monitor .apk. In Section 4.11 we discuss resources in detail.

Below we discuss components using the problem domain language, which might not always map directly to the object-oriented classes of the underlying implementation, yet the inconsistencies are minimal, if any.

4.6 Execution phases

Execution of DROIDMATE is organized into four phases. The *inlining* phase is stand-alone. *Initialization*, *processing apks*, and *wrap-up* are three subsequent phases of a proper DROIDMATE run. DROIDMATE reads all its inputs close to the beginning of the initialization phase and does most of the interesting work in the apks processing phase. It produces exploration log for each application

immediately after it is done exploring it. Diagnostic logs are output during all the phases. Finally, close to the end of wrap-up phase, it outputs an aggregated human-readable logs for manual analysis, including exploration summaries and charts.

Description of execution phases follows:

Inline To enable monitoring of the Android framework API method calls made by the AUE during exploration, the AUE .apk file has to be modified by process called *inlining* before it is given as input to DROIDMATE exploration. Conceptually, inlining takes as input a directory containing a set of .apk files to inline and outputs the inlined .apk files to appropriate output directory. Inlining is a part of the *monitoring component* and is described in detail in Section 4.9 and Section 4.11.

From the implementation standpoint, inlining is done during a stand-alone DROIDMATE run with appropriate command line flag making it inline the .apk files instead of conducting proper exploration. The command line arguments also have to give the path to a directory that contains the input apks and to a directory that will contain the inlined apks. Some of the DROIDMATE functionality for the initialize and wrap-up phases, described below, is also executed during the inlining phase, like e.g. initializing the diagnostic logging and reading the .apk files from the file system.

Initialize Initialization is the first phase of proper DROIDMATE execution. During this phase DROIDMATE reads all its inputs, starting with parsing command line arguments. DROIDMATE accepts 48 different command line arguments. Conceptually, most of them can be split into two categories: (a) paths to directories with various inputs and outputs (like .apk files and generated logs) and (b) knobs to fine-tune the *exploration* (Section 4.7), like how often to reset it or which random seed to use.

After parsing the arguments into `Configuration` class, DROIDMATE constructs an instance of the `ExploreCommand` class and executes it. The command first reads the set of input apks from directory whose path was parsed into `Configuration`. Next, it sets up the Android device on which the AUEs will be explored. The setup phase initializes the *monitoring component* (Section 4.9) and the *GUI automation component* (Section 4.8). When this is done, DROIDMATE is ready to start processing the apps contained in the input .apk files.

Process apks The input apks are now processed sequentially. For each .apk

file, the app contained within it is installed on the device via ADB (Android Debug Bridge), which is an executable that is a part of the Android SDK distribution. Next, the app is explored in an *exploration loop* from the *exploration component*, to be finally uninstalled and have its exploration data serialized to the hard disk in form of an exploration log. The exploration loop is the heart of DROIDMATE and is described in detail in Section 4.7.

If exploration of any given app fails, DROIDMATE will make an attempt to restart it. If it doesn't help, DROIDMATE saves any results obtained thus far and proceeds with exploring the next apps in queue. If no exploration of any app can continue (because e.g. the Android device has been disconnected) DROIDMATE detects that and gracefully terminates.

Wrap-up After processing of all the apks finishes, the device is cleaned up from the parts of the monitoring component and the GUI automation component. Next, the exploration logs are used to produce detailed human-readable reports composed of textual exploration summaries, charts and textual tabular data from which the charts were generated. The reports detail how many unique API calls have been seen, how many GUI interactions happened during each app exploration and so on. After the report is generated, DROIDMATE proceeds to produce final diagnostic logs: Any exceptions that have been gathered during exploration are logged plus the diagnostic summary log of DROIDMATE run is output.

4.7 Exploration component

Exploration is the heart of DROIDMATE. After the AUE has been successfully installed on the device, DROIDMATE executes *reset exploration action* which launches main activity of the AUE. The main activity is the component of AUE that gets launched when the user presses the app icon in the apps list. Its identifier can be obtained from the .apk file with the AAPT (Android Asset Packaging Tool) tool from Android SDK. DROIDMATE then operates in a loop, as seen on Figure 4.1, whose centerpiece is the *exploration strategy*, as described in detail in Subsection 4.7.1.

The exploration strategy is called for the first time with a result of conducting a *reset exploration action*. It takes as input *exploration action run result* which contains two elements that pertain to the device state after last *exploration action* was conducted:

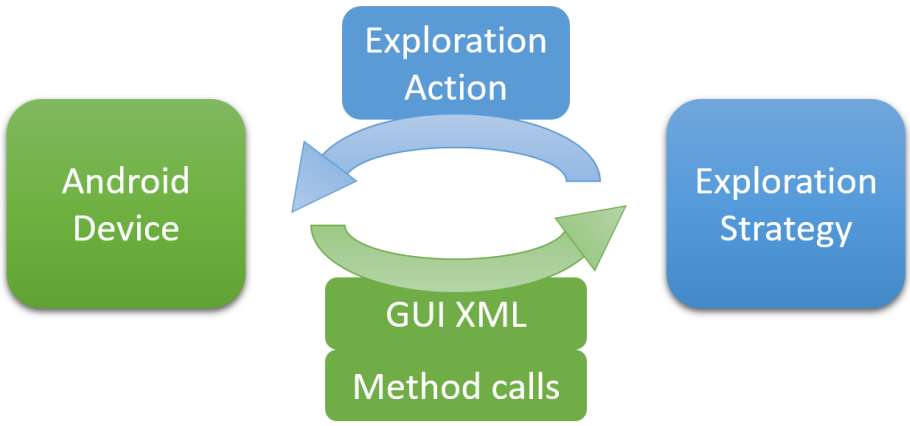

Figure 4.1: The exploration loop

- Information on the displayed GUI XML window hierarchy, wrapped in a GUI snapshot object. The window hierarchy is composed of Android's `View` objects [70].
- Logs obtained from monitored calls to Android framework API methods.

Given this information, exploration strategy *decides* which next exploration action to conduct. Exploration action can be any of: *click, long-click, reset, terminate*. The actions are then executed on the device, handling and recovering from a plethora of possible failures, up to and including intermittent loss of connection to the device. Exploration continues until some *termination criterion* is met, like time limit or limit on the count of exploration actions conducted.

The data output from the exploration is persisted as serialized Java objects, called *exploration logs*. DROIDMATE can produce various textual reports and charts from the serialized data. The data is very detailed and sufficient to replay the exploration, either manually or automatically.

4.7.1 Exploration strategy

Algorithm 1 represents exploration strategy of DROIDMATE. Exploration strategy operates on a high abstraction level, taking as input a *GUI state* and returning an *exploration action*. The GUI state contains an abstract representation of the GUI, hiding all the implementation details irrelevant for deciding what to explore next. It is extracted from GUI snapshot, which in turn is a wrapper

Algorithm 1 Exploration strategy.

Require: GUI State S
Ensure: Exploration action A

```
1: procedure DECIDE(S)
2:     if TERMINATE(S) then
3:         A ← terminate exploration
4:     else
5:         if RESET(S) then
6:             A ← reset exploration
7:         else
8:             A ←EXPLOREFORWARD(S)
9:         end if
10:    end if
11:    UPDATEINTERNALSTATE(S,A)
12:    return A;
13: end procedure
14:
15: procedure EXPLOREFORWARD(S)
16:    C ← view context of S
17:    VS ← views in C with minimal number of interactions so far
18:    V ← pick at random from VS
19:    A ← choose interaction action with V
20:    UPDATEKNOWNVIEWCONTEXTS(C)
21:    UPDATEINTERACTIONSCOUNT(V,C)
22:    return A
23: end procedure
```

over XML window hierarchy dump obtained from the device via UIAUTOMATOR, a GUI automation framework from Android SDK [101]. The exploration action in turn is an abstract representation of a possibly multi-step operation on the Android device like *click, long-click, reset* or *terminate.* DROIDMATE then translates this abstract representation into actual operations on the device, executes them, reads the resulting GUI state and API calls logs, and returns control to the exploration strategy.

The exploration strategy takes as input API calls made, in addition to the GUI state. They are both wrapped together in *exploration action run result*, as mentioned earlier. In current implementation, though, we do not leverage the

API calls information to decide which exploration action to conduct next. Thus, we omit it in Algorithm 1 and in further discussion.

Exploration actions are composed of many operations including reading device state, possibly multiple times, and manipulating the device, possibly making multiple attempts at it. The *click* and *long-click* exploration actions, in addition to multiple state reading operations (to obtain XML window hierarchy and saved API call logs), conduct simple device manipulations: They just execute one *GUI action*, directly mapped to the UIAUTOMATOR API, as described in Section 4.8. *Reset exploration action* is more complex and can execute multiple *press home* GUI actions, *turn Wi-Fi on* action as well as use ADB to execute *clear app package* and *launch main activity* operations. Its goal is to bring the device to a "clean slate" state and relaunch the AUE. *Terminate exploration action* just clears the installed AUE by clearing its package via ADB.

The exploration strategy implemented in DROIDMATE is inspired by Dynodroid [46]. The key idea is to *interact* with *views* [70] (i.e. GUI elements, also called *widgets*) randomly, but give precedence to views that have been so far interacted with the least amount of times. If multiple views have been interacted with minimal amount of times, we pick one randomly. A view interaction is either a click or a long-click (2 seconds). Interaction can happen only with views that are *enabled* as well as *clickable*, *long-clickable*, or *checkable*, as given by the attributes of XML window hierarchy obtained via UIAUTOMATOR (Section 4.8).

Each view is considered unique in its given *context* (also known as *widget context*)—that is, within the set of views that can be interacted with and appear on the same screen.

Thus, if a view appears in different contexts (i.e., surrounded by different GUI elements), it will be explored again in each of them. Contexts are different if they differ by at least one view. A view can differ by its fully qualified *class name,* its *resource ID* (if any), its *content description* (if any) and the rectangle describing its location on the screen. It can also differ by its *label,* unless the view's class has `<Switch>` or `<Toggle>` in its name.

Views can be *black-listed* if they lead to uninteresting situations. This includes the AUE crashing, launching a different app (like Google Play Store or Settings) or in any other way getting out of scope. A black-listed widget will not be interacted with again. All such situations result in DROIDMATE resetting the exploration. In addition, the strategy can be configured to reset the AUE exploration every given number of interactions. This makes it avoid getting stuck exploring a small subset of the app's GUI.

If, after launch, there are no views that can be interacted with, the exploration strategy resets the exploration. If, after reset, there are still no views, the

exploration of current app terminates. The exploration also terminates when the termination criterion is met, i.e. when the configured time limit or actions limit is reached.

4.8 GUI automation component

DROIDMATE GUI automation component is responsible for interacting with the Android device GUI, allowing execution of all the exploration actions as listed in Subsection 4.7.1. It also allows inspection of properties of GUI elements of currently displayed screen on the device, enabling the executed exploration actions to yield current GUI snapshot. The operations conducted with this component are done via two means: ADB and UIAUTOMATOR [101], the official Android GUI automation framework.

UIAUTOMATOR is implemented on top of JUnit, adding API that enables manipulation of the device GUI, providing methods to issue commands like e.g. "click on a visible button with label *send SMS*". We call such commands *GUI actions*. GUI actions are constituents of exploration actions. In addition, UIAUTOMATOR API allows for retrieving XML window hierarchy, which serves as a basis of GUI snapshot.

To use UIAUTOMATOR, developer is expected to write a UIAUTOMATOR JUnit test having a predefined sequence of steps to be conducted on the GUI when the test is executed. The tests are written on the developer machine. Next, they are packed as .jar file in case of Android 4.4.2 and as .apk file in Android ≥ 6 and higher. In both cases they are pushed/installed on the Android machine using ADB. Next, appropriate ADB command is issued to make the tests start executing.

Unfortunately, the standard scenario of using UIAUTOMATOR is not what is required by DROIDMATE. The input generator cannot just execute a sequence of GUI actions and be done with it. Instead, the generator has to read GUI snapshot after each exploration action. This GUI snapshot, possibly together with monitored API call logs, is used by exploration strategy to determine which next exploration action to execute. The sequence of commands to be executed is not known up-front. To accommodate for that, DROIDMATE requires for the UIAUTOMATOR test to be able to await commands from the *host machine*, i.e. the machine on which most of DROIDMATE code runs (the remaining code runs on the connected Android device). The test also has to send back the XML window hierarchy dumps when requested by the host machine.

We implemented the capability of awaiting commands by introducing the no-

tion of *UIAUTOMATOR-daemon*. The UIAUTOMATOR-daemon is a UIAUTOMATOR test that has TCP server running, listening to commands from the host machine. It is capable of receiving commands via TCP from the host like *click* and *long-click* on specific screen coordinates, *turn Wi-Fi on*, *press home* and similar. It can also send back via TCP the XML window hierarchy dump, which will be wrapped in the host machine into a GUI snapshot.

All the received commands are executed by making calls to the UIAUTOMATOR API. One exception is the Wi-Fi manipulation, which in case of Android 6 is executed through appropriate Android system service instead of UIAUTOMATOR API calls. In case of Android 4.4.2 the daemon just simulates the actions user would do on the GUI to manually turn on the Wi-Fi.

During the initialization phase of DROIDMATE (Section 4.6) first the .jar/.apk with the daemon is pushed to the device via ADB, then started. The host machine also forwards appropriate TCP ports using ADB. Next, the TCP server of the daemon signals it is ready by outputting appropriate message to logcat, Android's logging mechanism. As soon as host reads the message (also via ADB), it establishes TCP connection with the server and is ready to send commands and receive output like XML window hierarchy dumps. The TCP server is terminated during wrap-up phase of DROIDMATE after it received appropriate command from host.

Not all exploration action steps can be executed via UIAUTOMATOR-daemon. All the remaining steps, like *launch main activity* or *reset all app data* are done with ADB, without any assistance from UIAUTOMATOR. AAPT tool is used to extract AUE metadata to obtain the name of main activity to be launched.

4.9 Monitoring component

DROIDMATE monitoring component is responsible for logging the set of called Android framework API methods after each exploration action was conducted on the device and returning these logs to the exploration strategy. The logs contain the full method signature, parameter names and values, stack trace and thread ID. The monitoring component works by intercepting API method calls on Android device and redirecting them to code that logs all the data about given call before the call is actually executed (based on AppGuard [10], as explained below). The logs are then retrieved by the host machine using a TCP client/server pair. The monitoring component is composed of following parts:

Inlined AUE A rebuilt and resigned with debug key AUE which has its byte-

code modified. The bytecode has its `Application` class modified in a way making the AUE load `Monitor` class from an .apk present on the device on AUE startup (usually first launch of its main activity).

Monitor A resource whose centerpiece is a `Monitor` class, loaded by the inlined AUE on startup. Monitor contains definitions of which Android framework API methods have to be redirected to enable logging their calls, plus it contains TCP server for sending these logs back to the host machine. The monitor has to be pushed to the device in an .apk file, so it can be found and loaded on the inlined AUE startup. As the monitor is independent of the inlined .apk, it can be easily rebuilt with new set of methods to monitor: They are defined in a .txt file easy to edit by a human, described in more detail in Section 4.11. The monitor is rebuilt on each full DROIDMATE build, as discussed in detail in the same section.

TCP server The TCP server runs from the `Monitor` class, one instance for each process spawned by the AUE. The TCP servers are being asked by a TCP client located on a host machine at appropriate times while executing exploration actions to retrieve the API calls logged thus far.

TCP client Located on the host machine. Sending and receiving endpoint of commands and data directed to/from TCP servers located on the device. Used to gather API call logs.

During *inlining* phase (Section 4.6) DROIDMATE ensures the input .apk is inlined and thus, ready to load `Monitor` on the AUE startup. Next, in initialization phase, DROIDMATE pushes to the device an .apk file with `Monitor` class Dalvik bytecode, so it will be found and loaded by the AUE. As soon as the AUE .apk is installed and the app launched via ADB during the *processing apks* phase of DROIDMATE, the inlined bytecode finds and loads the monitor, reads its method redirection declarations and starts logging calls to the redirected methods, recording the logs in internal field. The `Monitor` class also launches TCP server, listening for any commands from the host machine, most importantly listening for requests to send the so far logged API calls data. Each of the processes spawned by the AUE has its own instance of `Monitor` class and thus server, all of which are queried by the host machine. During *wrap-up* phase DROIDMATE uninstalls the .apk, also stopping all the TCP servers.

The inlining of apks is done with *inliner*, a component of *inline reference monitor system* extracted from AppGuard [10]. The capability of redirecting (intercepting) Android framework API method calls is implemented in different

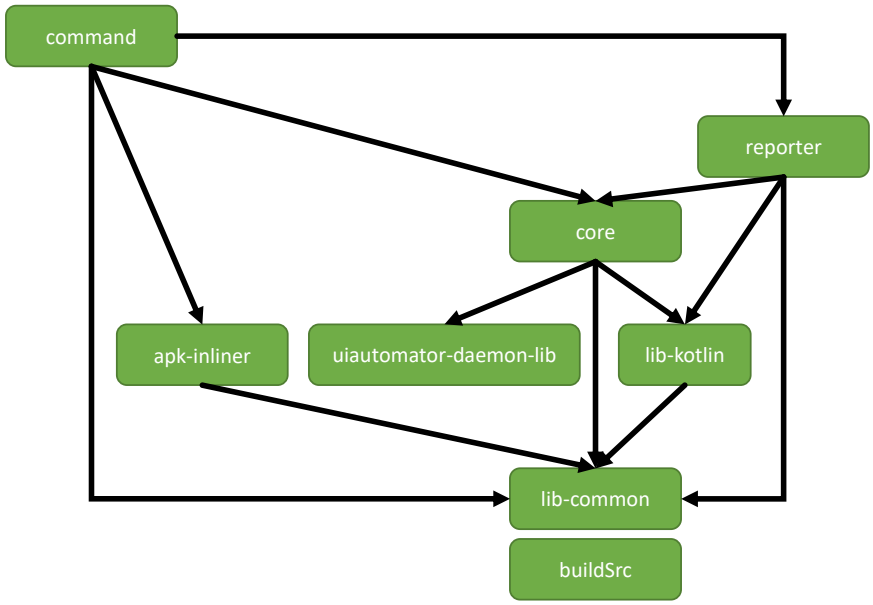

Figure 4.2: Compile-time dependencies of DROIDMATE modules. `buildSrc` is compiled first and most other modules depend on it.

ways depending on the Android OS version. On Android 4.4.2 it is done with another component of AppGuard, the *instrumentation component* [10]. On Android 6 and newer the same system cannot be used because Android runtime has changed to ART. Instead, we use ArtHook [72] framework. The differences are encapsulated in the `Monitor` class and .apk that contains it. There are two versions of the .apk file, each incorporating API and library with the required redirection/instrumentation framework. DROIDMATE uses appropriate one during its run, as configured by a command line argument. The process of building these two monitor apks is described in more detail in Section 4.11.

4.10 Modules

The modules of DROIDMATE and their compile-time dependencies are given in Figure 4.2. Each DROIDMATE module is in fact a *Gradle project*, that is, it is built with a `build.gradle` build script residing in the module's root direc-

tory. Entire DROIDMATE is built with Gradle, as explained in the DROIDMATE documentation [103].

The *command* module is the entry point of DROIDMATE, i.e. it contains the `main` method. The `main` method sets up logging, parses command line arguments into configuration and runs one of the available commands, handling any exceptions thrown. The commands are *inline*, *explore* or *report*. *Inline command* corresponds to the *Inline execution phase* described in Section 4.6; *explore command* runs all the remaining three phases; and the *report command* runs only a subset of the *wrap-up phase* that produces a DROIDMATE run report.

The *core* module contains bulk of DROIDMATE logic, including the build logic in its `build.gradle` script. The module internal structure is elaborated on in Subsection 4.10.1.

The *apk-inliner* module exposes the `ApkInliner` class which takes as an input a normal .apk file and outputs its inlined version, as required by the monitoring component described in Section 4.9. The module exposes a programmatic API, but it can be also called as a stand-alone program.

The *reporter* is used to generate a DROIDMATE run report containing all the human-readable text files and charts produced from the exploration log.

The *uiautomator-daemon-lib* provides serializable classes sent via TCP between the the host machine and the UIAUTOMATOR-daemon, located on the device.

The *lib-common* module is an infrastructure module. That is, it contains many classes that are used ubiquitously in DROIDMATE. Thus, it serves similar purpose to the infrastructure layer of the core module, as explained in Subsection 4.10.1. Yet lib-common is even lower level, i.e. the core module infrastructural layer depends on it. In addition, lib-common exists to be used by *all* other modules, not only core or modules dependent on core.

The *lib-kotlin* module serves the same purpose as lib-common module, but it is implemented in the Kotlin programming language. It has to be separate from lib-common to avoid potential problems caused by mixing multiple languages in the same module.

The *buildSrc* is even lower level module than *lib-common*. BuildSrc is special in the sense it gets prebuilt with Gradle before any other module gets built. This is because it contains values that have to be shared between modules and Gradle build scripts (`build.gradle` files) that build these modules.

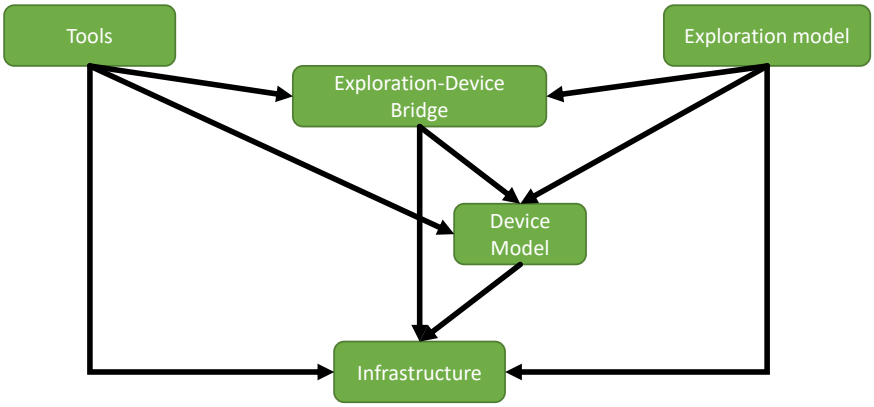

Figure 4.3: Compile-time dependencies of DROIDMATE's *core* module conceptual layers. Each layer represents a set of JVM classes organized into packages. There is a dependency between layers if any of the classes in given layer depends on any of the classes in the other layer.

4.10.1 Core module

Conceptually, core is split into five layers: *tools*, *exploration model*, *exploration-device bridge*, *device model* and *infrastructure*. The dependencies between the layers are given in Figure 4.3.

The *infrastructure layer* provides utility classes for all other layers in the core module and in some cases for the command and reporter modules.

The *tools layer* provides high-level abstractions enabling working with Android devices. This includes a class to obtain object representation of .apk files, a class to setup and tear-down properly setup Android device, a tool to do the same with an .apk file on that device, and a factory creating instances of `AndroidDevice` class.

The *exploration model layer* contains the domain model of the exploration, modeling concepts like exploration action, exploration strategy or exploration output.

The *exploration-device bridge layer* bridges access to the device model by adding additional capabilities increasing robustness and abstracting away the details of reading logs from the device. The main abstraction is provides is `RobustDevice` class, decorating `AndroidDevice` class, making it resistant and able to recover from AUE crashes and device crashes, among others.

The *device model layer* models the device. The most important class is `AndroidDevice`, which is a façade used by the host machine to communicate with the device. Other relevant classes are: `UiautomatorWindowDump`, representing the Android device GUI state, and `AndroidDeviceAction`, a class implementing the exploration action from exploration model layer in the domain of Android.

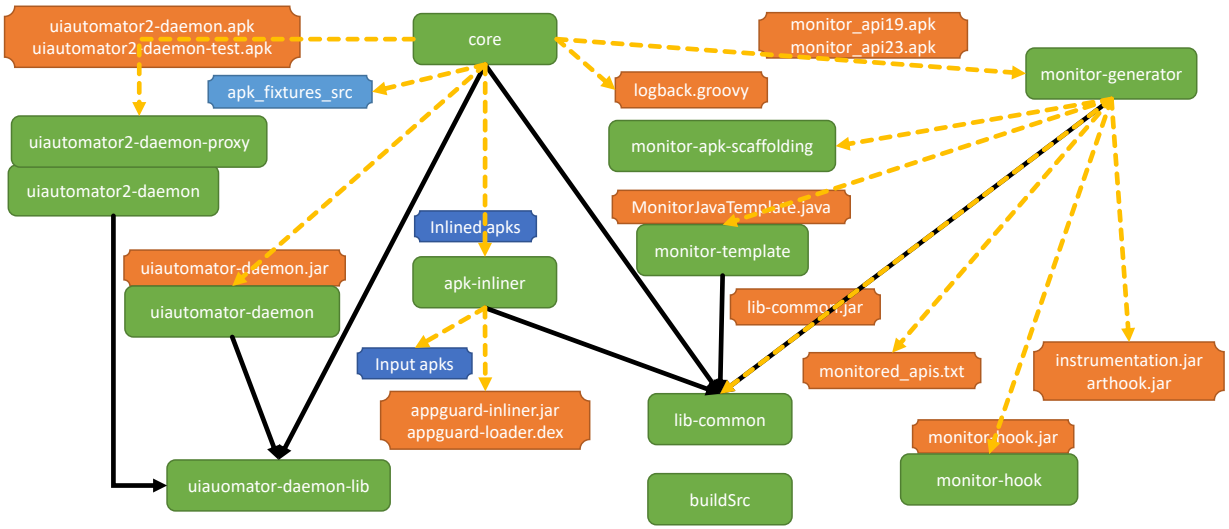

Figure 4.4: Resource dependencies of DROIDMATE modules. Solid arrows denote compile-time dependencies. Dashed arrows denote resource dependencies. Green rectangles denote DROIDMATE modules. Rectangles with clipped corners denote resources. `apk_fixtures_src` is a set of .apk fixtures for testing built by a separate Gradle project. `Input apks` have to be provided, they do not come with DROIDMATE source. `buildSrc` is compiled first and most other modules depend on it.

4.11 Resources

Key part to understanding inner workings of DROIDMATE is what *resources* it uses and how they are obtained. DROIDMATE requires resources to run in addition to inputs.

The names of the resources used by DROIDMATE and how they are processed during build by supporting modules are given in Figure 4.4. The diagram mentions several modules not discussed previously in Section 4.10. These modules exist solely to produce relevant resources and are explained below, while discussing the resources produced.

Logback is a logging framework leveraged by DROIDMATE. `logback.groovy` is its configuration file, setting up routing of output logs to various files.

DROIDMATE consumes as input directory with *inlined apks*, which are raw *input apks* inlined (injected) with bytecode of the apk-inliner. A stand-alone project, `apk_fixtures_src`, contains a set of inlined apk fixtures used for DROIDMATE end-to-end regression testing.

`appguard-inliner.jar` and `appguard-loader.dex` are parts of the AppGuard [10] inliner component, extracted from it and adapted to work with DROIDMATE. During inlining, the inliner bytecode payload is intertwined into the AUE .apk file to enable monitoring of the API calls. The inliner bytecode payload living in the .apk file loads and calls at runtime the DROIDMATE `Monitor` bytecode.

To enable IDE-supported development, the device-deployed `Monitor` class is constructed from a template file, `MonitorJavaTemplate.java`. The class depends at runtime on types from `lib-common.jar`. The `Monitor` has a compiled-in list of Android API methods to monitor generated from easily editable file, `monitored_apis.txt`. The monitoring logic is extensible: one can extend it by editing the monitor-hook module. The underlying library to natively interface with Android OS to record the monitored method calls including arguments, return values etc., is `instrumentation.jar` in case of Android 4.4.2 (API 19) and `arthook.jar` in case of Android API 23.

Everything is wired together by the monitor-generator module and inserted into the monitor-apk-scaffolding project. Depending on which framework was used, either `monitor_api19.apk` or `monitor_api23.apk` is output. While the entire process is complex due to various extensibility points and development convenience, the output .jar is built in one automated step.

Separately from the inlined .apk, and `monitor_*.apk`, an UIAUTOMATOR-daemon is deployed to the device to read the XML window hierarchy of the GUI during exploration and to execute GUI actions. The daemon does both of these

by calling into UIAUTOMATOR framework. Depending on the Android version, two different variants of the daemon are deployed, resulting in different .jar files. The daemon shares some types with the core module via the uiautomator-daemon-lib module.

4.11.1 Monitored API methods list

The complete list of monitored API methods used in experiments described in Chapter 6 is given in Appendix B. This list pertains to Android 4.4.2 (API 19). The API list is based on the list used internally by AppGuard. However, we had to spend significant manual effort into sanitizing it, removing redundant methods (i.e. methods just delegating to other monitored methods) and adding missing methods, as the list was outdated and incomplete. We analyzed the Android API documentation to find out missing methods from the groups of relevant monitored APIs, which have been added since the AppGuard list we used was obtained. The set of modifications for Android API 19 and 23 had to be different. There are some methods that are present in Android 4.4.2 but not 6, and vice versa. The `monitored_apis.txt` file has annotations understood by the monitor-generator to ensure right set of methods is monitored for the right Android OS version.

4.12 Limitations

DROIDMATE has several limitations:

- To enable monitoring of API method calls, the AUE has to have its bytecode modified and it needs to be resigned with a debug key. As the built-in Google apps like Maps or Contacts cannot be modified, DROIDMATE cannot obtain API call logs from them. Furthermore, If the app does remote integrity check on startup, it will fail, most likely just stopping upon startup.
- As exploration strategy depends on the XML window hierarchy of the explored app, it will not work on games and other apps using native development kit to render screen contents, like maps. Such apps have no meaningful XML representation of their screen.
- ArtHook is compatible only with ARM-based architectures, meaning if one wants to use DROIDMATE on Android 6 and be able to monitor calls

to API methods, one is forced to either use real ARM-based Android device or unacceptably slow ARM-based emulator. The fast emulators are Intel-based and thus incompatible with ArtHook.

- Due to limitations of the the monitoring component used for Android 4.4.2, DROIDMATE cannot monitor system calls made from native code [10] when running on this Android version. ArtHook, used with Android 6, can monitor such calls.
- The instrumentation framework used for monitoring API calls on Android 4.4.2 cannot monitor some non-native methods, most notably methods of `SmsManager`. Fortunately, this limitation is not present in ArtHook.
- Any application that requires login credentials needs to have additional, manually provided code that will guide DROIDMATE to login with credentials which were previously manually provided. Otherwise DROIDMATE will be limited to exploring only the functionality available without logging.
- DROIDMATE at this point has only partially implemented support for inputting text into text fields. We do not use it in this thesis.
- DROIDMATE can click or long-click, but doesn't support other gestures, like swiping.
- DROIDMATE cannot send any system events, e.g. *SMS received* or *Boot completed*.

Chapter 5

Sandboxing implementation: BoxMate

Chapter 4 described our realization of an *automated input generator*, DROIDMATE, which is one of the three components required to answer research questions posed in Section 3.7. This chapter discusses the remaining two components: A *sandbox policy inference engine* and a *sandbox*. All these three components together form full implementation of the *sandbox mining* concept which we call BOXMATE.

Our realization of the sandbox policy inference engine leverages DROIDMATE's ability to monitor Android framework API method calls. We propose two kinds of behavior policies: The *API call enforcement policy* and *event-bound API call enforcement policy*. They are described in Section 5.2 and Section 5.3 respectively. We made two proof of concept implementations of the sandbox component of BOXMATE: One based on AppGuard and one on Boxify, both discussed in Section 5.4.

5.1 Android permission model

BOXMATE introduces its own sandbox limiting Android apps behavior. However, one should ask what is the protection guaranteed by Android itself in the absence of BOXMATE and why it isn't enough. Indeed, Android apps are already sandboxed by the operation system. According to the official Android documentation on system permissions [75]:

> "Android is a privilege-separated operating system, in which each application runs with a distinct system identity (Linux user ID and group ID). Parts of the system are also separated into distinct identities. Linux thereby isolates applications from each other and from the system.
>
> Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad hoc access to specific pieces of data."

In addition, the security architecture section [85] of the same document states:

> "A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another application's files, performing network access, keeping the device awake, and so on.
>
> Because each Android application operates in a process sandbox, applications must explicitly share resources and data. They do this by declaring the permissions they need for additional capabilities not provided by the basic sandbox. Applications statically declare the permissions they require, and the Android system prompts the user for consent."

Comprehensive description of Android applications security is given in [86].

Practically speaking, Android app developer will have to declare permissions required by the app in its *app manifest* XML file [65] and the end-user will have to allow these permission when using the app. The permissions, in turn, guard Android framework API method calls. If an application tries to call an API method without having appropriate permission to access it, most of the time a `SecurityException` is raised [84]. Unfortunately, it is not completely clear which API methods require which permissions. Finding exact mapping between the permissions and API methods has been a subject of extensive studies [24, 7].

All *normal permissions*, as listed in [87], are always granted automatically by the OS when requested in the app manifest [88]. Granting these permissions poses little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission.

Dangerous permissions, as listed in [83], on the other hand, require user confirmation. These permissions are assigned to *permission groups*. Up until Android 5 (API 22), the user was presented with a list of permission groups to endorse at install time. For example, if the app manifest declared it requires `READ_CONTACTS` permission, the user was informed she has to grant access to `CONTACTS`, i.e. she was presented with the name of an entire permission group [83].

Since Android 6 (API 23), the permissions are no longer granted at install time, but at runtime. According to documentation describing Android 6 [83]:

> "If an app requests a dangerous permission listed in its manifest, and the app does not currently have any permissions in the permission group, the system shows a dialog box to the user describing the permission group that the app wants access to. The dialog box does not describe the specific permission within that group. For example, if an app requests the `READ_CONTACTS` permission, the system dialog box just says the app needs access to the device's contacts. If the user grants approval, the system gives the app just the permission it requested.
>
> If an app requests a dangerous permission listed in its manifest, and the app already has another dangerous permission in the same permission group, the system immediately grants the permission without any interaction with the user. For example, if an app had previously requested and been granted the `READ_CONTACTS` permission, and it then requests `WRITE_CONTACTS`, the system immediately grants that permission."

5.1.1 Android permission model is ineffective

The permission system is designed to be understandable to the user and help prevent attacks, such as social engineering attacks trying to convince device users to install malware [89]. In practice, however, it doesn't work well. In a survey of 308 Android users, Felt et al. [25] found that only 17% paid attention to permissions during installation (as seen on Android 5 or lower), and only 3% of all respondents could correctly answer three questions regarding permissions. While the new permission model doesn't ask for permissions at install time, one can observe that many Android apps reduce the new permission model to the old one: As soon as the user launches an app, a set of dialog boxes appears, one

per each permission group, asking the user to endorse all permission groups. As a consequence, the permission model, even in its new incarnation, is ineffective.

One could argue users ignore the permissions because they are *too generic* and are *all or nothing propositions*. Consider an application that requires registration. After you register, you receive SMS message to confirm you gave valid phone number during registration. The application reads the message automatically, to streamline the registration process. In addition, you can let the app send SMS messages to paid numbers to unlock additional features. In such scenario it would be natural for the user to allow the app to receive the authentication SMS message, but forbid sending any paid messages in the future. Unfortunately, the user is given the *all or nothing* proposition: Either block access to SMS, making it impossible to even login into the application, or allow the app to send, receive and read SMS messages, as dictated by the SMS permission group [83]. All these actions can happen at any time, from/to any number and also in the background, without user knowledge. Other permission groups suffer from the same problem. Because of the genericness of the permission groups, users possibly have learned that rejecting them will cripple the app to the point of being useless.

Let us now assume the user is well versed in the permission system and understands that rejecting given permission group doesn't have to cripple some apps, just disable some secondary features. Even so, because many permissions are grouped together, and the permissions themselves are quite coarse-grained, just allowing one critical permission group would allow the application to wreak havoc if it is a malware pretending to be a benign app. Consider a case in which you installed a project management application into which you want to import contact data of a small group of people involved in the project. Turns out the `CONTACTS` permission group forces you to allow the app to read, write and delete all contacts. If the application is indeed malware, it can easily steal all your contacts (`INTERNET` is a normal permission and thus allowed automatically [90]) or even delete them! In other words, the permissions are *too coarse-grained* resulting in too large attack surface.

Given the shortcomings of the Android permission model, for our BOXMATE implementation of the sandbox mining concept we propose new API-call based security policies that aim at being more fine-grained and more meaningful, thus hopefully getting proper user attention and significantly reducing attack surface. The policies are described in Section 5.2 and Section 5.3.

5.2 API call policy

As discussed in Section 5.1, built-in Android permission model is ineffective and thus unfit for the sandbox mining concept. Instead of limiting ourselves to groups of coarse-grained permissions, we propose our first behavior enforcement policy: The *API call enforcement policy*. Sandbox using this policy operates not on a level of permission groups, but on a level of Android framework API method signatures. If during the mining phase AUE will make a call to `getLastKnownLocation(String provider)` but will never make a call to `getGpsStatus(GpsStatus status)` then the first API call will be allowed at runtime during normal usage but the second one will be blocked or will require user confirmation, as per sandbox mining concept. If we would use Android permission model, we would have to allow the app to access the `LOCATION` permission group, giving unrestricted access to all methods of `LocationManager` [91], including the two just mentioned. Not only would it be ineffective, it would prevent us from getting valuable, detailed information about the AUE behavior based on the permissions it requests.

5.2.1 Distinguishing API calls

As explained in Chapter 4, we designed our input generator DROIDMATE to enable monitoring Android framework API method calls by leveraging parts of AppGuard. Because it is unknown which methods exactly require which permissions, we had to determine the set of methods to monitor. As AppGuard itself is a tool designed to give security and privacy-conscious end-users control over which *sensitive API calls* are allowed, we reused the sensitive API methods list used internally by AppGuard. After sanitizing the list and filling in the gaps we ended up with 97 sensitive API methods. We discuss how we did it in detail in Subsection 4.11.1. The full list of the used monitored API methods is given in Appendix B. It pertains to Android 4.4.2 (API 19), as this was the most recent version of Android at the time we conducted the experiments described in Chapter 6.

For each call of a monitored API method, DROIDMATE records:

1. The fully qualified name of the API method called, including class and method name and parameter and return types;

2. The thread ID and the entire thread call stack trace of the API call (starting at `Thread.run()` or Dalvik's native `main()`);

3. All arguments (actual runtime values of parameters), including security-relevant ones, like e.g. `ContentResolver` resource URIs;

4. Properties of the *triggering view*, if any. The properties are displayed text, associated resource ID, screen bounds, etc.

Most Android resources are uniquely identified by their specific set of Android framework API methods. Parameter values of these method calls can be ignored in most cases as they determine irrelevant details. For example, a call to `LocationManager.requestLocationUpdates(listener)` determines which listener to inform when a location has changed. Yet we are interested only if appropriate call to `LocationManager` was made at all, nit the listener.

However, one set of Android API methods heavily depends on the parameter values to identify the correct resource accessed. These methods pertain to *ContentResolvers*—that is, Android equivalents of databases. Knowing only that `ContentResolver.query()` was called is not enough, as the query may relate to all kinds of sensitive resources. For `ContentResolver` calls, DROIDMATE therefore also monitors the URI identifying the exact database, e.g. `content://com.android.contacts/data/phones`.
Sometimes, URIs end with the numeric identifier of particular instance of the resource being accessed: We consider all API calls differing only by this identifier as equivalent.

Triggering view association

The *triggering view* is the GUI element which has been clicked or long-clicked when conducting given exploration action on the AUE. Of course, in case of exploration actions that reset or terminate, no such view is present. The API calls made are *associated* with the view based on time: When the click or long-click happens, DROIDMATE waits until GUI finishes updating and then reads through TCP from the device all the API calls that have been recorded by the monitor. In case there is no view present, the API calls are associated to the reset or terminate exploration action in the same way.

API calls equivalence

We can now say that if during exploration two Android framework API calls take place, we consider them *distinct API method calls* if and only if the called API methods signatures differ. The only exception are calls to `ContentResolver`

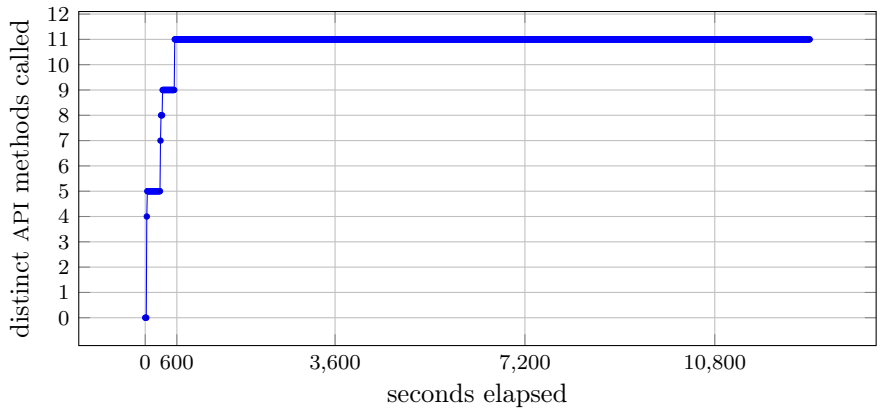

Figure 5.1: DROIDMATE API call saturation of SNAPCHAT exploration. After 10 minutes (600 seconds), DROIDMATE has made 11 distinct calls to API methods used by SNAPCHAT.

methods, in which the calls are considered distinct not only if the signature differs, but also if the value of the URI parameter differs.

5.2.2 A Snapchat case study

As an example of how DROIDMATE explores application behavior, let us consider the SNAPCHAT application. Figure 5.1 lists the number of unique API calls discovered during testing. As we discover more calls, we say that *API call saturation* occurs. This means we are getting closer to making all possible distinct API method calls we can potentially make with given exploration strategy, which is determined by the input generator. The actual distinct API method calls (in order of discovery) are listed in Figure 5.2, including the identifiers of the GUI elements that triggered them:

APIs 1-4 After a click on the `login_button` on the start view, SNAPCHAT opens a socket (API 1) which allows establishing a connection to a HTTP server. It also opens the camera (API 2), queries the current location (API 3) and accesses account info via a URL connection (API 4).

API 5 Taking a picture (`camera_take_snap_button`) starts monitoring the current location.

APIs 6-7 Recording a video sets the video and audio sources for recording, initializing the media recorder.

API 8 Later, DROIDMATE finds the SNAPCHAT "My friends" button (the unlabeled element), which requires accesses to the image library.

API 9 SNAPCHAT allows for finding friends based on their phone number, requiring access to contacts.

API 10 Saving a picture stores it to a database.

API 11 Previewing a snap deletes it after the preview is done.

```
[Button com.snapchat.android:id/login_button]
   1  java.net.Socket: void <init>
   2  android.hardware.Camera.open()
   3  android.location.LocationManager.getLastKnownLocation()
   4  java.net.URLConnection openConnection()
[Button com.snapchat.android:id/camera_take_snap_button]
   5  android.location.LocationManager.isProviderEnabled()
[Button com.snapchat.android:id/camera_take_snap_button
(long-click)]
   6  android.media.MediaRecorder.setAudioSource()
   7  android.media.MediaRecorder.setVideoSource()
[unlabeled GUI element]
   8  android.content.ContentResolver.query()
         uri = content://media/external/images/media
[Button com.snapchat.android:id/contacts_permission_button]
   9  android.content.ContentResolver.query()
         uri = content://com.android.contacts/data/phones
[ImageButton com.snapchat.android:id/picture_save_pic]
  10  android.content.ContentResolver.insert()
         uri = content://media/external/images/media
[RelativeLayout com.snapchat.android:id/
snap_preview_relative_layout]
  11  android.content.ContentResolver.delete()
         uri = content://media/external/images/media/<number>
```

Figure 5.2: The 11 SNAPCHAT calls to sensitive API methods discovered by DROIDMATE, and the events (in `[]`) that first triggered them.

The mined method calls provide a behavior description seen during the mining phase of sandbox mining, later codified as behavior enforcement policy. No other distinct API calls will be allowed by a sandbox in production than the calls observed in the mining phase. Other distinct calls will require explicit, contextualized consent of the user. Ideally, the 11 distinct API method calls

mined with DROIDMATE should encompass all common, expected behavior, and only it. This way it will be rare for a new, yet unseen distinct API method call, to happen. Thus, the user burden on making the decision to allow given call or not will be manageable. At the same time, the user will be presented with very specific information about the new fine-grained behaviors, avoiding both flaws of Android permission model, i.e. genericness and the "all or nothing" dilemma.

5.3 Event-bound API calls policy

By default, BOXMATE simply checks whether the app as a whole uses the same distinct API methods as recorded and distinguished during exploration; we call this *API call enforcement policy*. This policy allows for quick saturation during mining, and thus few false alarms during enforcement (sandboxing phase); however, it may be too coarse to prevent some attacks. For instance, once we have seen that SNAPCHAT can read contact phone numbers, *any function* within SNAPCHAT, including background tasks, would be allowed to do that. However, as we have seen in Figure 5.2, SNAPCHAT accesses phone numbers only to allow the user to find other SNAPCHAT users among her friends. How about restricting contact access to this functionality only?

To this end, BOXMATE implements a *more fine-grained* enforcement policy. During sandboxing, *event-bound API call enforcement policy* also verifies whether the API call was triggered by *the same event* as during mining:

1. In the mining phase, during exploration, BOXMATE records *(event, API call)* pairs, where *event* is the *identifier of the event-triggering GUI element* if present, or special identifier *reset* or *terminate*, depending on the exploration action. The *API call* in turn is a call to a sensitive API method *associated* with the event. The association works for all events as for *triggering views*, as explained in Section 5.2.1. We call such pairs *event-bound API calls*.

2. During sandboxing, upon each call to a monitored API method, *API call'*, triggered by an *event'*, BOXMATE checks whether *(event', API call')* pair was already found during mining; if not, the call is *flagged*. A flagged call is either outright blocked or presented to the end-user, with contextualized, human-readable information. The user has then to endorse or forbid the call and all calls of such type in the future.

Since our "events" are primarily interactions with named GUI elements, and as our API calls all refer to user-owned resources, the BOXMATE event-bound

enforcement policy realizes the principle of *User-Driven Access Control* [54, 56], namely, by tying access to *user-owned resources* to *user actions* in the context of an application.

5.3.1 Distinguishing events

BOXMATE applies the following rules to identify events. All views (GUI elements) in Android [70] have three features:

- A *resource identifier* r that associates views and programmatic actions (`<login_button>`);
- A *text label* l possibly displayed on the screen ("Login");
- A *content description* d that can be read out loud to the user as an accessibility feature ("Login").

While most of these features are defined in an XML layout file, all of them can also be defined or changed at runtime; hence the need for a dynamic analysis.

BOXMATE stores an event e as a tuple $e = (id, action)$:

id by default is the resource identifier r; if r is empty, $id = d$ instead; and if d is empty as well, $id = l$ instead. We prefer identifiers to labels since the latter may change during operation—for instance when changing the app's language.

action is the *user interaction* that triggers the event; for widgets (i.e. interactive views like buttons), this is either a click or a long-click.

With these rules, two widgets are different even if they sport the same text ("ok"), as long as they have different resource identifiers.

The following rules apply for special events:

- If all of r, l, and d are empty, e has the special value *unlabeled*. All *unlabeled* events are treated as one, as there is no easy way of determining which two unlabeled events are the same or different.
- If the thread ID is not equal to 1 (the GUI thread), e has the special value *background*. Again, all *background* events are treated as one.
- If the app is reset (restarted) or exploration is terminated, e has the special value *reset*. This captures events occurring during exploration start and end, among others.

5.4 Sandboxes

The last element of BOXMATE, after an input generator and behavior enforcement policy inference engine, is the sandbox. The sandbox has to be resistant to tampering, otherwise a hacker could circumvent it, completely defeating the guarantees provided by sandbox mining. The sandbox also has to be performant at analyzing each API call or *(event, API call)* pair, otherwise it will make the app being secured unusable. While in this thesis we focus on input generation and the policy inference, we also implemented a minimal proof-of-concept of a sandbox by reusing fragments of two technologies: AppGuard and Boxify. Both of them are described in the following section, including their flaws and advantages.

An example of an early prototype of the sandbox showing a sandbox policy violation to the user is shown on Figure 5.3. The dialog box look and feel is the same and independent of the sandbox implementation used: AppGuard or Boxify.

5.4.1 AppGuard

AppGuard by Backes et al. [10] is a framework for securing untrusted Android applications which builds upon the concept of *inline reference monitoring* (IRM). The key idea of IRM is to rewrite an untrusted application such that the reference monitor that enforces the security policy is embedded directly into the untrusted app's code. To this end, AppGuard takes an untrusted app and user-defined security policies as input and produces a secured self-monitoring app which can run on unmodified Android devices, even without root access.

Technically, AppGuard diverts control flow towards the security monitor by modifying references to the monitored, security-relevant methods in the Dalvik Virtual Machine's internal bytecode representation. More precisely, by modifying key data structures, such as virtual method tables, in the VM's memory at runtime, AppGuard can efficiently intercept calls to sensitive API method calls from Java code.

AppGuard has several limitations:

- To enable method interception, the app under monitoring needs to have minimal change to its bytecode made, making sure it loads the monitor and starts monitoring the declared methods on startup. While the changes to the app bytecode are minimal, this technique still modifies the app

signature and requires resigning with non-genuine developer key, and thus will not work on apps doing developer signature integrity checks.

- AppGuard can monitor Java methods invoked from native code, but it cannot monitor system calls made from native code.
- As the AppGuard monitor initialization code is baked directly into the apps' bytecode, a malicious app might try to detect AppGuard modification and tamper with the monitor through reflection. While AppGuard provides means to thwart such attacks by restricting access to the reflection facilities of Java, a sufficiently determined attacker will still be able to disable the monitor.
- In some cases AppGuard monitoring performance overhead rises up to 21%. In practice, however, this is rare and any runtime impact is not noticeable by the user.

We used parts of AppGuard in DROIDMATE to monitor calls to sensitive Android framework API methods. Fortunately, the same element can be used for sandboxing: Instead of just logging a call to an API method, we can also compare it with defined enforcement policy and if violation appears, show a pop-up dialog box asking user for decision, remember it, and block or let the call happen. Thus, while AppGuard suffers from some limitations, extending BOXMATE to provide an experimental proof-of-concept AppGuard-based sandbox was a relatively easy task. In the prototype we did not implement the more sophisticated AppGuard features, such as its automata-based security policies, its protection against forceful extraction of stored secrets, or its interactive interface. These features could easily be added by integrating full AppGuard into BOXMATE.

5.4.2 Boxify

Due to the flaws of AppGuard listed in previous section we also experimented with another sandbox implementation based on Boxify, also by Backes et al. [9]. Boxify is a novel approach for Android application sandboxing, which provides tamper-protected reference monitoring for stock Android without the need for root privileges. Boxify uses app virtualization and process-based privilege separation to encapsulate untrusted applications in a restricted execution environment within the context of another, trusted sandbox application. To establish a restricted execution environment, Boxify leverages Android's *isolated process*

feature, which allows apps to completely de-privilege selected components. By loading untrusted apps into de-privileged, isolated processes, Boxify avoids modifying apps and provides strong security guarantees, unlike AppGuard. Sensitive I/O operations are relayed through a separate, privileged broker process which monitors and enforces policies.

Boxify checks whether the API call is allowed by the mined sandbox rules; if not, it can either have the call return a mock object containing fake data, or flag the call, asking the user for permission, as seen on Figure 5.3. Only calls to the monitored sensitive methods incur any overhead, which is 1–12% per call [9], resulting in practically no runtime performance overhead overall.

The main drawback of Boxify is that a sandbox implementation based on it is significantly more involved and depends on internal Android APIs. Such APIs are very likely to change with each Android version, thus increasing the maintenance effort of keeping Boxify-based sandbox up-to-date with the newest Android OS.

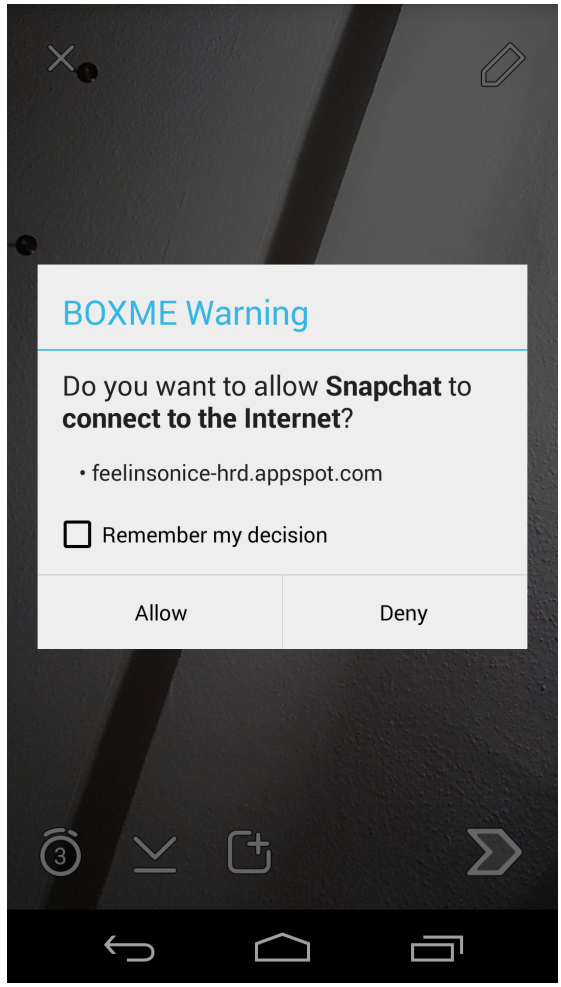

Figure 5.3: The BOXMATE sandbox in action. Calling a sensitive API method not seen during mining requires confirmation by the user. To facilitate readability, API names are automatically mapped to the respective Android permissions, which are then shown in user-readable form.

Chapter 6

Sandbox quality study

In this chapter we use BOXMATE, discussed in Chapter 5, to answer the first three out of four research questions posed in Section 3.7. The questions are:

Q1 *Can input generators sufficiently cover behavior?*

Q2 *Can we reduce the attack surface by providing a behavior enforcement policy more fine-grained than Android permission system?*

Q3 *Can the more fine-grained mined behavior help users and experts correctly classify behavior as benign or malicious?*

We will answer the fourth question in a study described in Chapter 7.

To answer Q1, we will mine real world Android applications for their behaviors and codify them in two behavior enforcement policies, one more fine-grained than the other. We will then evaluate how often sandboxes enforcing these two policies flag yet unseen behaviors when applied to real world usage scenarios of the same Android apps. In other words, the scenarios will be our *ground truth*. We will also analyze how long it took for a behavior saturation to occur during the mining. If the policies can be mined in reasonable time, i.e. saturation is quick, and sandbox violations turn out to be rare, it will mean the behavior was sufficiently covered by the exploration done with input generator used in the mining phase. Thus, the answer to Q1 will be affirmative.

To answer Q2, we will assess if the two policies inferred from the mined behavior are more restrictive than the Android permission system, thus reducing attack surface.

Name	Version	Category	Rank	Identifier
Adobe Reader	11.1.3	Productivity	1	com.adobe.reader
AntiVirus Security – FREE	3.6	Communication	5	com.antivirus
Barcode & QR Scanner barcoo	3.6	Shopping	6	de.barcoo.android
CleanMaster – Free Optimizer	5.1.0	Tool	1	com.cleanmaster.security
Currency converter	1.02	Finance	9	com.frank_weber.forex2
eBay	2.5.0.31	Shopping	1	com.ebay.mobile
ES Task Manager (Task Killer)	1.4.2	Business	10	com.estrongs.android.taskmanager
Expense Manager	2.2.3	Finance	24	at.markushi.expensemanager
File Manager (Explorer)	1.16.7	Business	1	com.rhmsoft.fm
Firefox Browser for Android	28.0.1	Communication	7	org.mozilla.firefox
Job Search	2.3	Business	6	com.indeed.android.jobsearch
PicsArt – Photo Studio	4.1.1	Photography	1	com.picsart.studio
Snapchat	4.1.07	Social	4	com.snapchat.android

Table 6.1: Evaluation subjects. The *Rank* column denotes the rank of given app in given category at the time of download, i.e. 25 March, 2014. Open `https://play.google.com/store/apps/details?id=`⟨Identifier⟩ for details.

To answer Q3, we will compare old and new version of the same Android application. We will compare the differences of policies inferred from behaviors mined from both versions and assess if they help the user determine if the new version is a legitimate update to an application or does it instead hide unexpected, possibly malicious, functionality.

6.1 Experimental setup

To answer the questions according to the plan outlined in previous section, we need multiple elements:

Android platform We will run all of this chapter experiments on Android platform version 4.4.2. We use older of the two versions supported by DROIDMATE (as explained in Section 4.4), as this was the most recent Android version at the time we were conducting experiments described in this chapter.

Apps evaluation set We will infer sandbox policies from behaviors mined from a set of 13 very popular Android applications downloaded from German Google Play Store, like SNAPCHAT or Adobe Reader. The metadata of the used apps is given in Table 6.1.

Two versions of same app To answer Q3, we will mine and compare two versions of SNAPCHAT: 4.1.07 and 5.0.34.6.

Input generator To generate inputs for exploration conducted in the mining phase, we will use DROIDMATE (Chapter 4), the input generator of BOXMATE.

Explorations In the mining phase, we will explore the apps from evaluation set to obtain exploration logs. The exploration logs will be used to produce saturation charts, exploration summaries, and to extract mined behaviors, which in turn will be used to infer behavior enforcement policies. Saturation charts and the inferred policies are described further down below.

Explorations time limit As we do not have means to determine if we observed entire possible behavior of given app, we cannot determine if we covered some percent of all possible behaviors. Instead, we will limit the time of explorations. For all the apps from the evaluation set except SNAPCHAT, we will run the explorations for 2 hours, hoping to observe behavior saturation, hinting we may be close to discovering all possible behaviors. As SNAPCHAT is our running example for which we want to obtain more thorough results, we will instead explore both of its versions for 3.5 hours.

Explorations reset frequency We set our explorations to reset every 30 interactions, to avoid getting stuck in a subset of the AUE GUI.

Behavior enforcement policies We will infer from the mined behaviors (extracted from the exploration logs) two behavior enforcement policies for the sandbox: The *API call enforcement policy* and *event-bound API call enforcement policy*. Analysis of the policies will let us know if they indeed reduce attack surface as compared to Android permission system, thus answering Q2.

Behavior saturation charts To determine if we have reached the exploration behavior discovery limit, we will plot behavior saturation charts from the exploration summaries, just like the chart in Figure 5.1. We will have two chart for each app: One per each policy used. One chart will plot the count of distinct Android framework API method calls discovered over time. The second chart will plot count of distinct *(event, API call)* pairs instead of just API method calls. If no new behaviors will be discovered close to the exploration time limit, i.e. the chart will be "flat", we will conclude

we reached saturation, i.e. behavior discovery limit. The limit might be actual limit of all possible behaviors, or input generator power limit, which may be lower, in case there are behaviors given input generator will never be able to discover.

Real world use cases (ground truth) We will compare the mined enforcement policies with real world behaviors of Android apps, determining how often the sandbox flags resource accesses, thus answering Q1. We represent the real world app behaviors by a set of 18 manually written use cases. The use cases are automated, UIAUTOMATOR-based tests, interacting with the apps' GUIs. The sequences of operations on the GUIs are representative of conducting real world common use cases, as derived from app descriptions. For example, for Adobe Reader, the most common use case is opening and viewing a .pdf document. We designed the use cases to cover as many secondary functionalities as possible and reasonable for given scenario. The use cases and functionalities they exercise are given in Table 6.2. Textual descriptions of the exact GUI operations can be found within Appendix C.

On average, implementing a single use case and having it replay reliably took us 2–3 hours of work. This perhaps surprisingly high implementation effort was due to implementation flaws of UIAUTOMATOR which required workarounds and due to general difficulty of hand-scripting user interactions. This observation additionally motivates the use of automated input generators such as DROIDMATE.

6.1.1 Evaluation plan

Our evaluation plan is as follows:

1. Conduct explorations with appropriate time limits, using GUI inputs from DROIDMATE, on the applications from the evaluation set. As a result, obtain exploration log for each app.

2. Derive two policies from mined behaviors extracted from the exploration logs: *API call enforcement policy* and *event-bound API call enforcement policy*.

[1]Despite our best efforts, neither we nor DROIDMATE could get barcoo 3.6 to use the camera and scan something on our devices.

App	Use Case	Functions
Adobe Reader	View Document	What's New, Help, Open first document
AntiVirus	Scan	Activate, Scan now, View scan results
barcoo	Search for product	Search "pillow" in search box, View results[1]
CleanMaster	Scan	Scan system, Resolve all, Report
Currency Cvtr	Convert currency	Enter "159", Swap currencies
eBay	Find by search	Accept terms, Sign in, Search "pillow", View first search result
ES Task Mgr	Kill task	Kill first listed task
Expense Mgr	Add and edit expense	Add an expense of $15.80 for "Pills" in Category "Health"
	Delete expense	Open history, Delete first entry
	View and set budget	Set a total budget of $7.00 in the "other" category
File Manager	View and create dir	View directories, create new directory "temp_utc"
Firefox	Open URL	Go to "google.com"
Job Search	Search for job	Search a job for "sales" in "New York, NY", Select first result
PicsArt	Apply effect	Apply "twilight" effect on recent photo, Save on SD card
Snapchat	Take snap	Log in, Take snap, Add caption, Set retention, Send snap to self, View it
	Take video	Log in, Take video, Pick color, Draw Line, Save to gallery, Add to story
	Find friend	Log in, Add friend from contacts, Allow Access
	Edit friend	Log in, Search friend "abc", Block "abc", Unblock "abc", Delete "abc"

Table 6.2: Use cases.

3. Using data from exploration logs, plot saturation charts of the recorded behaviors, taking into account the data which is required by the policies. Analyze the charts to see how quickly mined behaviors have saturated during the explorations.

4. Run the hand-written tests representing real world use cases and monitor the API accesses and GUI elements used, giving us exploration summaries representing the use cases. This is analogous to running normal explorations, but inputs instead of being generated with DROIDMATE are predefined, as they come from the hand-written tests.

5. Determine, based on the exploration summaries obtained from real world use cases, how often the mined policies would be violated (i.e. flagged by a sandbox) by the observed behaviors. Thus, answer Q1 and Q2.

6. Determine how clear for a human are the differences between the old and new version of SNAPCHAT, thus answering Q3. Obtain the list of differences by comparing mined behaviors from both versions.

6.2 Resource access saturation

We conducted explorations of the evaluation set, inferred the policies and plotted saturation charts, i.e. we executed steps 1, 2 and 3 of our evaluation plan (Subsection 6.1.1). The API call saturation charts for 2 hour explorations of twelve apps are seen on Figure 6.1. Figure 5.1 shows the 3.5 hour exploration of SNAPCHAT.

We see that ten charts "flatten" before one hour mark and the remaining two before two hours. For SNAPCHAT, we reach saturation in less than 10 minutes. Recall that sandbox mining has to happen once when the application is first installed and once every time a major update appears that requires access to new resources, to confirm the update didn't introduce any malicious behavior. As sandbox mining needs to be executed rarely, the obtained times are well within reasonable expectations.

The saturation of API calls is enough for the API call enforcement policy, but for the event-bound API call enforcement policy we have to pair the API calls with events that triggered them, as described in Section 5.3. Fortunately, the same exploration provides all necessary data. Saturation of event-bound API calls (*(event, API call)* pairs) of the 2 hour explorations of the twelve apps from evaluation set are visible on Figure 6.3. The 3.5 hour SNAPCHAT exploration saturation chart is given on Figure 6.2. In contrast to Figure 5.1, we see that it takes more than an hour of exploration of SNAPCHAT until the chart flattens at over 90% of all event-bound API calls ever explored. A similar late saturation can also be seen when mining event-bound API calls for the other twelve apps, as summarized in Figure 6.3.

In some cases the event-bound API call saturation didn't flatten in the allocated time. Namely, for Barcode & QR Scanner barcoo, CleanMaster, Currency converter, eBay and ES Task Manager. To understand the reason for that, we investigated the exploration summaries, available in full in Appendix C. In the case of Currency converter we are lead to believe we have actually seen all the possible pairs, as there are only two distinct API calls, `Socket.<init>` and `URL.openConnection()`, both bound to events `<reset>` and `background`. In all the remaining cases the lack of saturation was caused by spurious connection between API calls and events. The same API methods were called over and over, each time being bound to the GUI element which was acted upon last time. The API calls are most likely not related to the GUI elements at all. This is inaccuracy of our method of associating API calls by time to the last exploration action. More accurate association would likely solve this problem and flatten the charts.

Overall, while the times to reach event-bound API call saturation are significantly longer, they should be still within tolerance limit of security-conscious user willing to use such fine-grained policy. We conclude that:

Explorations with automatically generated inputs can quickly cover resource usage. However, more fine-grained analysis of the usage takes longer to mine.

We now know that we can get close to the limit of behaviors covered with automatically generated inputs in a reasonable amount of time. To fully answer Q1 we still need to determine if the covered behaviors are enough to represent common use cases and thus, avoid violating sandbox too often. This is discussed in next section.

6.3 Policy violations

To understand how often API call enforcement policy obtained with BOXMATE would flag real behaviors, we conducted steps 4 and 5 of our evaluation plan, i.e. checked the inferred policy against behaviors representing real world use cases. The count of violations observed is summarized in Table 6.3. Each violation results in a sandbox flagging the behavior, showing pop-up to the user asking her to endorse or block it.

In the case of "Search for Product" use case of QR Scanner barcoo app, an unlabeled not explored button requests the current location, violating sandbox enforcing the more fine-grained policy. CleanMaster requires user attention three times: Two for changes to configuration when a scan is started or a report is sent, and one when a handle to *PowerManager$WakeLock* is acquired after the scan is finished. Again, that happens only while using the more fine-grained policy. The eBay "Find by Search" use case requires login credentials, while we explicitly didn't gave them to DROIDMATE, forcing it to explore only the functionality available without logging into the app. The use case, in turn, explores GUI parts available only after logging, causing the need for confirmation. The PicsArt "Apply effect" use case accesses an existing photo from an SD card, which was not found during testing. In the SNAPCHAT "Take video" use case, a "status" button accesses the external media the video is saved in.

Overall, for the API call enforcement policy we end up with only 2 confirmations while executing 18 use cases. Not only the enforcement policy greatly reduces attack surface, the user attention is actually needed less times than with Android permission model, where the user would have to confirm access to more than 2 permission groups. The event-bound API call enforcement policy requires

App	Use Case	Flags per:	app	event
Adobe Reader	View Document		-	-
AntiVirus	Scan		-	-
barcoo	Search for product		-	1
CleanMaster	Scan		-	3
Currency Cvtr	Convert currency		-	-
eBay	Find by search		1	1
ES Task Mgr	Kill task		-	-
Expense Mgr	Add and edit expense		-	-
	Delete expense		-	-
	View and set budget		-	-
File Manager	View and create dir		-	-
Firefox	Open URL		-	-
Job Search	Search for job		-	-
PicsArt	Apply effect		1	2
Snapchat	Take snap		-	1
	Take video		-	-
	Find friend		-	-
	Edit friend		-	-
Total flags (out of 18 use cases)			2	8

Table 6.3: Use case actions flagged by sandbox with API call policy ("app" column) and event-bound API call policy ("event").

user input 8 times, but reduces attack surface even further and provides richer contextual information for the user to make the decision.

We deem the amount of sandbox violations negligible for the API call enforcement policy and very tolerable for the event-bound API call enforcement policy. This result, together with reasonable behavior saturation times during exploration, lets us answer questions Q1 and Q2 in an affirmative way:

***Q1A**: Input generators can cover behavior sufficiently fast to encompass primary use cases. The mined sandbox policies will impact application usability only in a minimal way.*

***Q2A**: The mined sandbox policies greatly reduce attack surface.*

6.4 Version differences evaluation

The pop-ups asking user to confirm BOXMATE policies have by design more relevant information than Android permission system, helping the user correctly recognize and block unwanted behaviors. A natural scenario in which we can evaluate Q3 is the case of an significant application update requesting more permissions. We conducted the final, sixth step of the evaluation plan (Subsection 6.1.1), by comparing API call enforcement policy of two significantly different versions of SNAPCHAT. In our primary evaluation, we explored SNAPCHAT version 4.1.07 for 2 hours. For this experiment we additionally explored SNAPCHAT version 5.0.34.6 for 2 hours. We then compared the inferred policies and highlighted the differences. Full exploration summaries are given in Appendix D. Saturation charts for both versions are given in Figure 6.4. Looking at the charts, we can see that SNAPCHAT 5 calls just 2 more API methods. However, the API methods called by the versions actually differ more. Overall, we found SNAPCHAT 5 makes seven API method calls not present in exploration of SNAPCHAT 4. That is, SNAPCHAT 5:

1. Uses Android 4.x `AudioRecord` interface, while the old version used the Android 1.x `MediaRecorder` interface instead.
2. Requires read and write access to image thumbnails through methods of `ContentResolver` interface like `query()`, `openFileDescriptor()` and `insert()` (3 distinct calls), while reading the SNAPCHAT privacy policy.
3. Accesses the user's `line1` phone number (after clicking on a button labeled `mobile_number`).
4. Uses the Android 4.x `PowerManager` interface, forcing the device screen to stay on while the message is sent (`send_to_bottom_panel_send_button` button).

While we did the comparison for the API call enforcement policy, we can leverage the information obtained from the event-bound API call enforcement policy to put the differences in the context of related GUI events. The most sensitive data, the user's phone number, is only accessed after the user has clicked on the appropriate button, acknowledging access. Just as we compared the respective sandboxes to determine what has changed in SNAPCHAT, any person could also have determined other changes between old and new versions of possibly less trustworthy programs.

A user wishing to preserve privacy settings could also run the untrusted SNAPCHAT 5 version within the trusted sandbox mined from SNAPCHAT 4, with any new API method calls being detected by the SNAPCHAT 4 sandbox. Then, she would have to confirm access once in each of the four situations:

1. When recording audio (for the `AudioRecord` interface),
2. When reading the SNAPCHAT privacy policy,
3. When setting phone number (for the `line1` phone number), and
4. When sending a message (for the `PowerManager` interface).

Each case would inform the user that there is a new feature—and thus enable her to detect, assess, and prevent potentially malicious behavior changes.[2] Our answer to **Q3** is thus positive:

> ***Q3A**: Mined policies can help in assessing and comparing app behavior.*

6.5 Threats and Limitations

Although our results demonstrate the principal feasibility of sandbox mining, we would not generalize our findings into external validity. Our sample of programs is small, is all on Android, and all GUI-based. For other programs and platforms, we may have to devise significantly different input generators, possibly requiring models of the program input structure as well as the sensitive resources to be monitored and protected. These input generators may be less successful in exploring program behavior, leading to more false alarms.

The set of use cases we have compiled for assessing the risk of unnecessary sandbox violations (Table 6.2) does not and cannot cover the entire range of functionality of the analyzed apps. While we assume that the listed use cases represent the most important functionality, other usage scenarios may yield different results.

Finally, keep in mind that in the absence of a specification, a mined policy can never express whether behavior is benign or malicious; and thus, our approach cannot eliminate the risks of both false alarms (sandbox violations on

[2] Note that whether the user sees these alarms as "false" entirely depends on the trust the user puts in the new SNAPCHAT version.

benign behavior) and missed attacks (malicious behaviors seen during mining and thus allowed). However, by detecting and preventing and bringing to the user attention unexpected changes, our approach is set to reduce both these risks, even in absence of specifications. On top of that, existing specifications for benign or malicious behavior could be merged with the automatically mined policies.

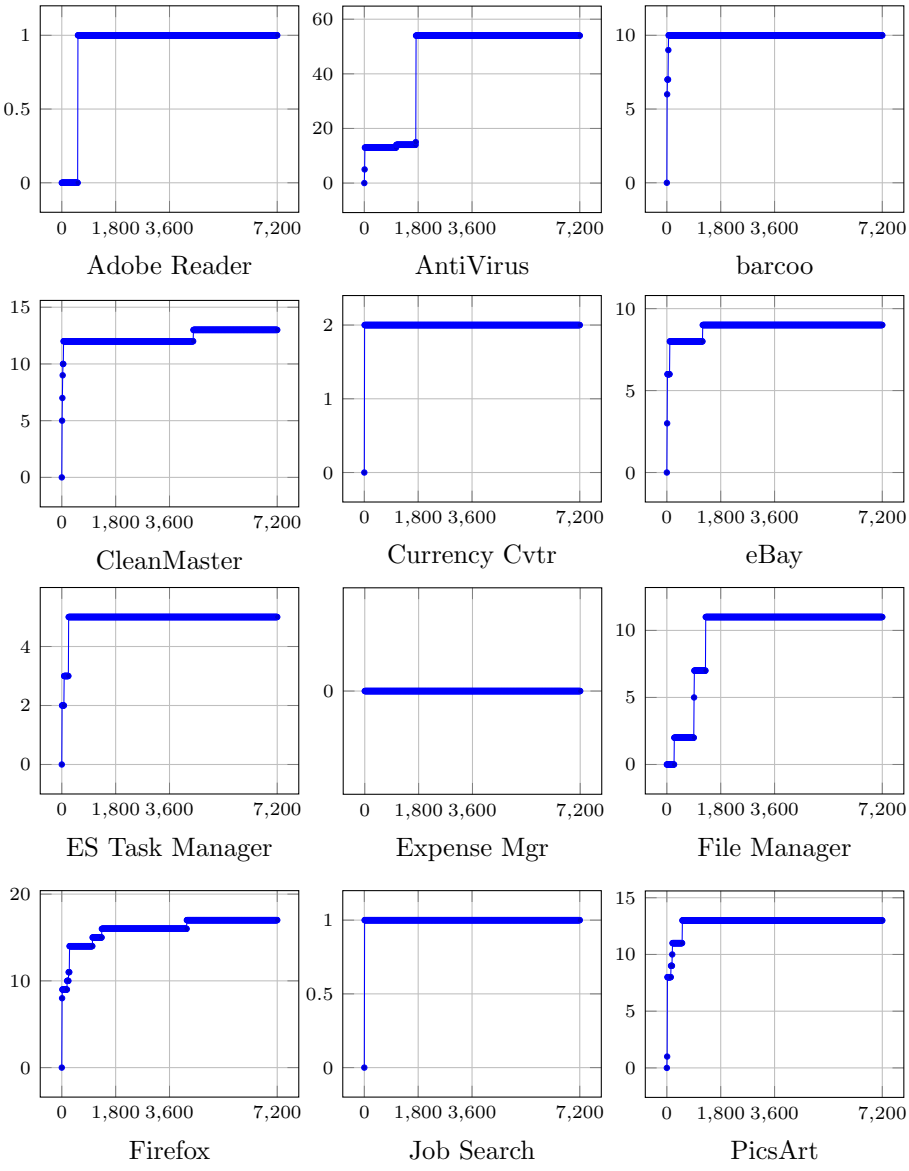

Figure 6.1: API call saturation of the apps in Table 6.1. As in Figure 5.1, the y axis is distinct API methods called; the x axis is seconds elapsed.

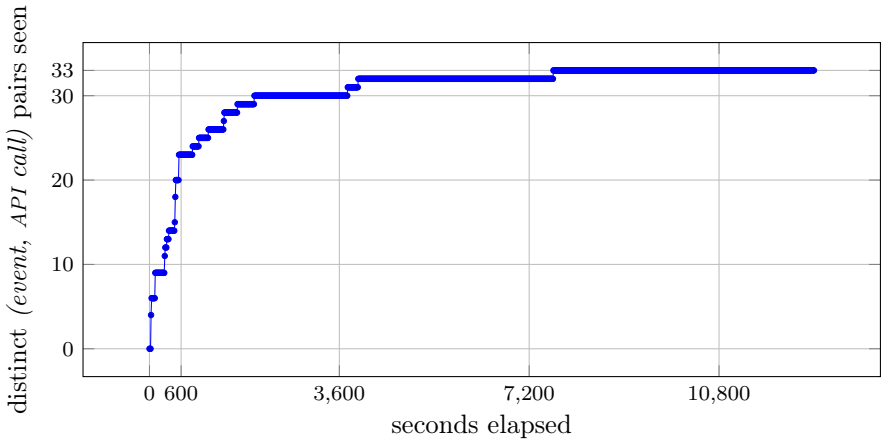

Figure 6.2: Event-bound API call saturation of SNAPCHAT exploration. After 60 minutes (3,600 seconds), DROIDMATE has discovered 30 unique *(event, API call)* pairs used by SNAPCHAT.

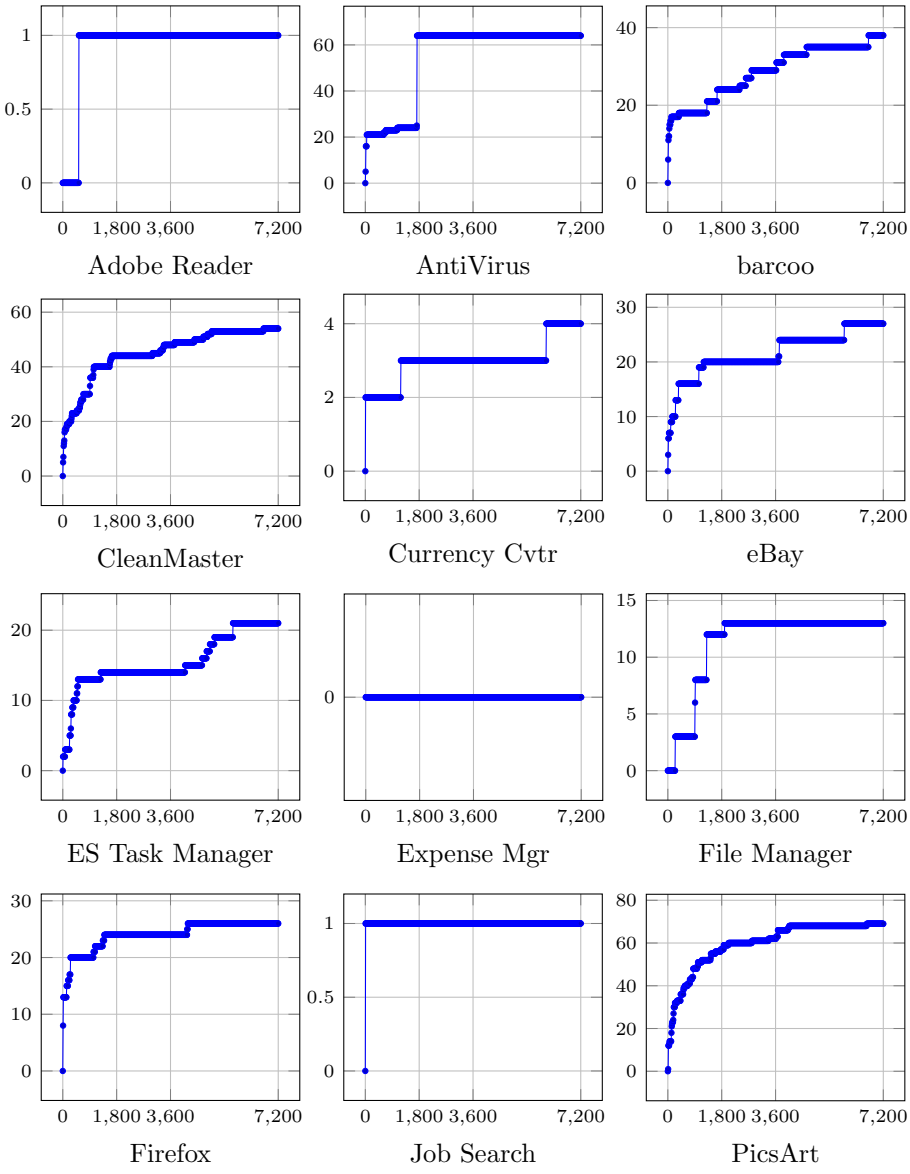

Figure 6.3: Event-bound API call saturation for the apps in Table 6.1. As in Figure 6.2, the y axis is distinct *(event, API call)* pairs seen; the x axis is seconds elapsed.

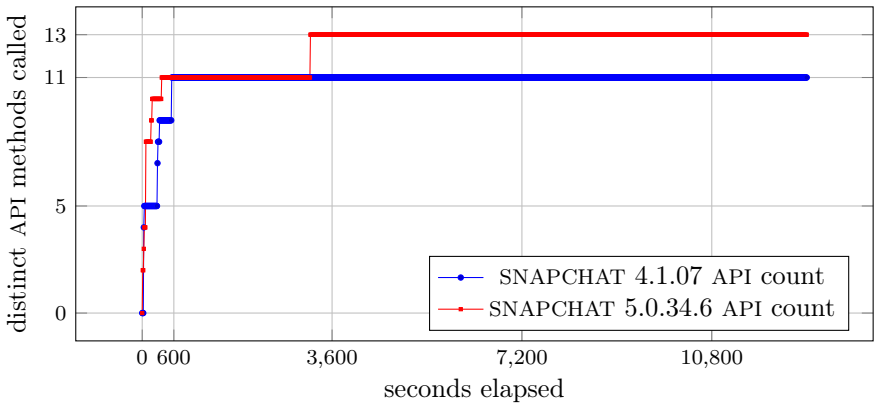

Figure 6.4: DROIDMATE API call saturation of compared the SNAPCHAT versions. The upper thin red line is the newer SNAPCHAT 5 version.

Chapter 7

Robustness study

In this section we describe our robustness study which we conducted to answer the final question posed in Section 3.7:

Q4 *Can modern input generators be successfully and fully automatically used to mine sandboxes from a variety of existing, widely used applications?*

Our sandbox quality study in Chapter 6 aimed to evaluate quality of the mined policies. It was conducted on manually preselected Android applications that were popular and didn't have any features that crippled exploration, like lack of GUI elements that can be explored, or integrity checks that prevented exploration from starting altogether. While the study has shown that indeed high quality behavior can be mined from some of the popular (and thus widely used) existing Android apps, it doesn't really tell us how good the mining works in general. In other words, it doesn't tell us if our test generator DROIDMATE is robust enough to handle majority of most popular applications. We conduct the following study to determine exactly that. Instead of manually preselecting applications, we run DROIDMATE on the top most popular applications from Google Play Store and report how well the exploration works: if it can be started at all, and if so, does it appear to explore meaningful behavior, or if it is crippled for any reason.

The evaluation plan for this study is simple: run DROIDMATE for a short period of time, i.e for 3 minutes, on the *top 1 apps* evaluation set, which is introduced in Section 7.2. Manually analyze the obtained exploration logs to determine exploration quality. Classify the explorations into categories, like: exploration doesn't start at all, exploration starts but is crippled, or exploration

works within the known exploration limitations of DROIDMATE as listed in Section 4.12. Diagnose and report the problems, reporting symptoms, root cause and possible future solutions.

7.1 Experimental setup

Analogously to equivalent section of previous chapter, Section 6.1, now we will describe what we require for our experimental setup to answer Q4.

Android platform We will run all of the robustness experiments on Android platform version 6, the newer of the two versions supported by DROIDMATE, as explained in Section 4.4.

Apps evaluation set Well-chosen app evaluation set is the most important element of this study. We decided to conduct explorations on the *top 1 apps*, that is, we picked the single most popular application for each of the 26 non-game categories in the Google Play Store. Details on the selection process and the apps chosen are given in Section 7.2.

Input generator As in previous chapter, to generate inputs for exploration conducted in the mining phase, we will use DROIDMATE (Chapter 4), the input generator of BOXMATE. This time, however, we will leverage ArtHook for runtime API call monitoring, as we are running on Android 6.

Explorations Like in previous study, we will need to obtain exploration logs from explorations. We will manually analyze the logs to categorize the apps by their robustness, as described in App exploration robustness analysis below.

Explorations time limit As we are only interested in determining if exploration is not crippled from the start, we do not explore for long. We will run each of the apps for 3 minutes. This will be enough for app robustness analysis, as explained below.

Explorations reset frequency We set our explorations to reset every 15 interactions. This is twice as often as in previous experiment, as described in Section 6.1. We do this to ensure we get varied behavior even with the short exploration time.

7.1.1 App exploration robustness analysis

We need to manually analyze exploration logs from each of the evaluated apps to determine given app robustness. We will report the *DROIDMATE exploration robustness category* of the app, *symptoms* of it, *root cause* of the symptoms and *possible DROIDMATE upgrade or other fix*, if any. To produce the report, we will manually look at:

- exploration summaries;
- saturation charts;
- charts showing the ratio of *views that have been interacted with* vs the *views that can be interacted with*, where a view interaction is defined in Subsection 4.7.1;
- the sequence of observed *widget contexts*, as defined in Subsection 4.7.1, with additional information on which widgets DROIDMATE decided to interact with and which of the views became black-listed;
- what the app displays as it is being explored;
- Android logcat output, to see if any Android platform runtime exceptions were thrown.

We will explore the apps for only 3 minutes each, but this will already should result in rich enough data to tell us if e.g. DROIDMATE became stuck and cannot explore much. Stuck exploration would manifest itself in immediately flattening charts that show the counts of views that can and were interacted with. It would also result in repeated encounters of the same widget contexts, with decisions to always interact with the same views.

7.2 Evaluation set

For this study to be valid, the apps for the evaluation set have to be chosen in an uniform way, not hand-picked. Furthermore, they have to be representative of complex, popular, real-world apps. This way the analyzed apps will serve as a good proxy of general DROIDMATE robustness. If it will work on the most popular, complex apps, it will likely work on majority of all apps.

To fulfill these goals, we introduce the *top 1 apps* evaluation set, with the concrete app data listed in Table 7.1. Each of the 26 apps is a representative of

the Google Play Store category to which it belongs. We consider the 26 categories which are not games, because due to DROIDMATE limitations, as described in Section 4.12, it cannot explore native GUIs well, which are predominant in games.

The app chosen for given category was the most popular app in that category, of *Top Free* kind (the only other kind, not considered, is *Top New Free*) on German Google Play Store in the time period from 22nd January 2014 to 26th May 2015. If there were multiple versions of the same app, the newest one was chosen, as determined by the `versionCode` property of the `details.json` metadata file crawled from the Google Play Store app data. Table 7.1 lists the date at which the app was at top of its category. If there were multiple such dates in the considered time period, one was picked at random. Also, in the time span, multiple apps from the same category might have been in the top 1 spot. If this was the case, we have chosen one from those apps at random.

7.3 Robustness results

The results of the experiments conducted on the evaluation set described in this section are given in Table 7.2. The third column, *Robustness*, describes *exploration robustness category* we assigned to given app based on our manual app robustness analysis of the experiment runs. The *Fix difficulty* column gives an estimate how difficult it would be to improve DROIDMATE to make exploration of given app robust, if at all possible.

Following subsections discuss the results in detail, explaining the robustness classes we identified and describing all the robustness problems we encountered, including symptoms, root causes and specific DROIDMATE improvements that would have to be applied to make the explorations robust. In next section we summarize the results.

7.3.1 Robust explorations

DROIDMATE is capable of automatically installing, starting and exploring apps classified as *OK* without any problems. In all such apps the exploration logs have shown that DROIDMATE was steadily calling new APIs, discovering new views to be interacted with and was steadily interacting for the first time with new widgets. However, of course the explorations were still subject to DROIDMATE limitations as described in Section 4.12. To give some examples of what DROIDMATE couldn't do:

MyTrails: clicking and swiping on a map;

DB Navigator: providing textual input for *from/to* destinations;

Earth: clicking and swiping on map; curiously, DROIDMATE was able to navigate the planet anyway by clicking on thumbnails of suggested places;

wetter.com: providing textual input for searched city name.

In spite of these limitations, there was lots of functionality that could be successfully explored.

7.3.2 Explorations requiring login

Explorations categorized as *requiring login* are just that: most of their functionality is blocked because they expect for the user to provide user name and password to an existing, registered account, and confirm by pressing appropriate button. While we list inability to login as one of DROIDMATE limitations in Section 4.12, we believe this limitation is serious enough that we cannot just claim explorations are robust in spite of requiring login. Yet, it is worthwhile to mention than **eBay** has significant amount of functionality explorable even without login, as it allows to browse through the auctions catalog.

We consider a fix for requiring login difficult. To achieve complete automation, the input generator would have to be able to fully automatically recognize that application requires login to registered account, find a way to register, register (likely confirming via email and defeating anti-bot measures like reCAPTCHA), and properly login with the registration information. One could settle for a partially manual approach. For example, register the account manually and provide the input generator with some hints how to login, or even make it capable of replaying manually recorded login sequence on each exploration reset. This solution however makes it impossible to apply explorations on massive scale for e.g. extracting data from many apps for analysis: One needs to manually register for every explored app, after all. However, for the envisioned goal of mining sandboxes of increasing app security, this might be acceptable manual effort. End-users mining sandboxes would have to register two accounts: one for their personal usage as they already do, and one for the sandbox mining, so it won't tamper with actual user data.

7.3.3 Stuck explorations

If analysis of logs determined that DROIDMATE is unable to discover and interact with new views, or, in other words, the amount of widgets that can be interacted with will saturate in the allocated 3 minutes, we concluded the exploration is *stuck*. We confirmed the behavior by additional manual inspection of how app behaved during the short exploration. We encountered two cases of stuck explorations: on the *Talking Angela* and *Daily News* apps.

In the case of **Talking Angela**, the symptoms are following: on startup, the app shows up a pop-up box stating "Get free gold coins for every push notification" with an "OK" button that DROIDMATE never clicks. Instead, DROIDMATE clicks repeatedly on 3 views that it detects based on the XML representation of the GUI returned from UIAUTOMATOR. However, these widgets cannot be seen, as they are hidden in the background that is blackened to display the pop-up. The underlying cause of the problem is that the pop-up box "OK" button is not marked as clickable, even though it is a `TextView` with resource id `push_button` and within current view bounds. Instead, its container with an id of `topSoftViewPlaceholder`, is marked as clickable, and indeed DROIDMATE tries to click on it (it is one of the 3 views on which it is stuck). However, the container bounds are equivalent to entire screen. Thus, DROIDMATE tries to click in the very center of entire screen, which is out of bounds of the "OK" button.

The solution to this problem would be for DROIDMATE to recognize that some views are clickable even if they themselves are not marked as such, only their containers. Easier, but less robust solution, would be to recognize pop-up boxes and/or buttons, by trying to match by ids (like `*button*` where `*` is any sequence of characters) or using some other heuristic. In any case, it is not a trivial problem, but doable with reasonable amount of effort. Implementing pattern matching for most common GUI conventions should cover majority of cases, and there are no significant technical obstacles to it. Given that the problem could be solved reasonably well with nontrivial but not prohibitive amount of effort, we deem the fix difficulty as *medium*.

The situation with **Daily News** app is much simpler: on startup, it immediately displays a pop-up with message "Network error, pull to refresh". DROIDMATE tries to repeatedly close it, yet it always reappears. We concluded this situation is caused by the app being defective. It behaved the same way when run completely manually, without any inlining applied to it. Its behavior might be caused by the fact it is not the newest version, and app developers might have adapted the server-side API to work with newer versions, breaking compatibility

with the one we used. This cannot be fixed from DROIDMATE side. The correct course of action here would be to download newer version of the app and see if it works, but this was beyond the scope of our experimental setup.

7.3.4 Explorations terminating early

Five application explorations terminated early: Google Now Launcher, Polaris Office, Runtastic, Facebook version 16 and Facebook version 28.

Google Now Launcher immediately displays a pop-up stating the launcher has to be set to default in Android settings to be used. There are two buttons: "Cancel" and "Settings". Both when clicked lead to a screen that is out of app scope, thus triggering app reset and becoming black-listed, as described in Subsection 4.7.1. As a result, on third reset, after black-listing both views that could be interacted with, DROIDMATE concludes there is nothing more it can do and terminates the exploration.

DROIDMATE improvement to handle such case would be rather nontrivial: it would have to recognize it has to take appropriate action in the Android Settings app, which it currently treats just as any other out-of-scope app. Implementing this would require giving Android Settings special treatment, understanding what can be set there, and what setting the explored app required to be changed. In this case, the default launcher app setting would have to be adjusted. Overall, we consider such extension of DROIDMATE difficult to realize, especially if it would have to work with various settings adjustment requests, not only setting default launcher app.

The case of **Polaris Office** app is simple: the app is defective. It crashes immediately after launch. Android's `ActivityManager` reports on logcat that app process has died. We reproduced the behavior when running the app manually, without any DROIDMATE involvement. The only course of action around this problem would be to try out newer version of the app.

Runtastic case is also simple: the XML window hierarchy of the screen visible after startup has no views that can be interacted with, just one large button with "SWIPE" label. Inability to swipe is a known DROIDMATE limitation as listed in Section 4.12, but because in this case it completely cripples the exploration, we classify it as "Terminates early" not "OK". Adding support for swiping to DROIDMATE would be highly nontrivial. Technically speaking, implementing the swiping functionality is a matter of making appropriate call to UIAUTOMATOR API. But swipes may happen in different directions, at various speeds, length and even via curved paths, not by straight lines. DROIDMATE would have understand which swipe is required in any given case, and that

would be the main difficulty in implementing this feature. Thus, we deem it difficult to implement this in a way that works robustly for majority of apps.

For **Facebook**, its version 16 has the same problem as version 28. DROIDMATE is too impatient: it thinks Facebook has finished loading, while it hasn't. This results in an empty screen with no views that can be explored. Normally DROIDMATE is able to detect such cases, as usually applications have some sort of progress bar, which periodically changes, making DROIDMATE recognize the application is still working and not ready for an interaction. However, in the case of facebook, nothing happens for over a minute. The simplest solution would be to make DROIDMATE always wait for a minute after reset, but that would lead to needless waste of time when exploring other apps. More intelligent solution would be to progressively extend wait times after reset. For example, first wait 10 seconds, then after first reset, 1 minute, then, after second reset, 5 minutes. Only if after waiting 5 minutes there still is nothing to be explored, finally abandon the exploration. Implementing this would be simple given current implementation, and thus we consider the fix to be easy.

7.3.5 No exploration

DROIDMATE is not even able to launch **Threema QR Code Plugin**. DROIDMATE launches applications either by directly calling their main activity via ADB, or by clicking on the app icon in the apps menu. This application doesn't have main activity nor does it add any icon to the apps menu upon install. We concluded the application is actually a plugin for another app. To make it work, DROIDMATE would have to be able to recognize the base application, i.e. the one to which the installed plugin pertains. Next, it would have to explore that base app, focusing on exploring the plugin functionality. This would be far beyond the current DROIDMATE implementation capabilities, and so we consider such fix to be difficult.

7.4 Results summary

13 out of 26, that is, exactly half of all the tested applications can have significant amount of their functionality explored fully automatically. We marked as difficult to fix 8 of the apps, i.e. 30.7%. They would require significant additional work to make their explorations robust. The remaining 5 apps, i.e. 19.2%, are either easy or medium to make robust, or it cannot be determined due to an app fault. These results denote DROIDMATE robustness is far from

perfect. However, the evaluated subjects are very complex and literally most popular apps in existence. In spite of that, even a research prototype grade input generator is able to automatically explore half of them, and with a reasonable amount of additional effort, could handle up to almost 70%. In principle, the exploration of remaining 30% is also possible, albeit not without additional significant implementation effort possibly leveraging nontrivial algorithms. Given these results, we consider fair to claim that an industrial-strength input generator would be able to explore thoroughly vast majority of applications. Thus, we conclude that the answer to our Q4 is positive:

> ***Q4A**: Modern, industrial-strength input generators should be able to successfully and fully automatically mine sandboxes from a variety of existing, widely used applications.*

7.5 Threats to validity

Similarly to the threats of our previous study, as described in Section 6.5, this study also focuses only on Android, GUI-based apps, that are not games. Even in the domain under our consideration, we have chosen only one application per each Google Play Store category. To mitigate this threat, we have chosen, for each category, the most popular app from a large time span. We hope such apps are fairly complex and can be considered an upper-bound of given category exploration difficulty.

We conducted only 3 minute long explorations. Ideally, we should explore until a saturation of observed APIs and interacted widgets is achieved, or it is concluded it cannot happen due to imprecise representation of GUI model by DROIDMATE. However, even in this short time span DROIDMATE was able to usually conduct more than 50 interactions, providing a good sample of app's behavior. To additionally mitigate this issue, we manually analyzed results from each of the apps and looked at the results from many different angles, as explained in Subsection 7.1.1.

Name	Version	Category	YMD	Identifier
WikiExplorer	1.5.5	Books & Reference	2014 7 9	animaonline.android.wikiexplorer
Polaris Office	6.0.9	Business	2015 2 25	com.infraware.office.link
Shqip TV	2	Comics	2014 2 2	arlind.Shqip
Messenger	12.0.0.21.14	Communication	2014 9 17	com.facebook.orca
Duolingo	2.7.2	Education	2014 8 21	com.duolingo
Talking Angela	2.2	Entertainment	2014 2 24	com.outfit7.talkingangelafree
PayPal	5.11.3	Finance	2015 3 10	com.paypal.android.p2pmobile
Runtastic	5.2.1	Health & Fitness	2014 9 3	com.runtastic.android
Threema QR Code Plugin	1.1	Libraries & Demo	2014 9 11	ch.threema.qrscannerplugin
eBay Kleinanzeigen	5.0.3	Lifestyle	2014 8 19	com.ebay.kleinanzeigen
PicsArt	4.6.12	Live Wallpaper	2014 10 15	com.picsart.studio
VLC	0.9.9	Media & Video	2014 9 6	org.videolan.vlc.betav7neon
Lady Pill Reminder	2.1.2	Medical	2014 8 16	com.baviux.pillreminder
Spotify	1.4.0.631	Music & Audio	2014 9 4	com.spotify.music
Daily News	1.13	News & Magazines	2014 7 3	com.ng.dailynews
Zedge	4.10.2	Personalization	2015 5 9	net.zedge.android
A Better Camera	3.24	Photography	2014 8 2	com.almalence.opencam
Adobe Reader	11.5.0.1	Productivity	2014 8 30	com.adobe.reader
eBay	2.8.2.1	Shopping	2014 11 20	com.ebay.mobile
Facebook	28.0.0.20.16	Social	2015 3 6	com.facebook.katana
MyTrails	1.4.5	Sports	2014 3 14	com.frogsparks.mytrails
Google Now Launcher	1.1.0.1167994	Tools	2014 10 25	com.google.android.launcher
DB Navigator	15.04.p06.00	Transportation	2015 4 17	de.hafas.android.db
Earth	7.1.3.1255	Travel & Local	2014 8 2	com.google.earth
wetter.com	2.3.1	Weather	2015 5 22	com.wetter.androidclient
Facebook	16.0.0.20.15	Widgets	2014 8 30	com.facebook.katana

Table 7.1: Robustness study evaluation subjects, sorted alphabetically by Category. The *YMD* column denotes the year, month and day of the date at which given app was the top 1 free app in given category at the German Google Play Store.
Open `https://play.google.com/store/apps/details?id=`⟨Identifier⟩ for details.

Name	Play category	Robustness	Fix difficulty
WikiExplorer	Books & Reference	OK	N/A
Shqip TV	Comics	OK	N/A
eBay Kleinanzeigen	Lifestyle	OK	N/A
PicsArt	Live Wallpaper	OK	N/A
VLC	Media & Video	OK	N/A
Lady Pill Reminder	Medical	OK	N/A
Zedge	Personalization	OK	N/A
A Better Camera	Photography	OK	N/A
Adobe Reader	Productivity	OK	N/A
MyTrails	Sports	OK	N/A
DB Navigator	Transportation	OK	N/A
Earth	Travel & Local	OK	N/A
wetter.com	Weather	OK	N/A
PayPal	Finance	Login required	Difficult
eBay	Shopping	Login required	Difficult
Messenger	Communication	Login required	Difficult
Duolingo	Education	Login required	Difficult
Spotify	Music & Audio	Login required	Difficult
Talking Angela	Entertainment	Gets stuck	Medium
Daily News	News & Magazines	Gets stuck	N/A
Google Now Launcher	Tools	Terminates early	Difficult
Polaris Office	Business	Terminates early	N/A
Runtastic	Health & Fitness	Terminates early	Difficult
Facebook v28	Social	Terminates early	Easy
Facebook v16	Widgets	Terminates early	Easy
Threema QR Code Plugin	Libraries & Demo	Cannot launch	Difficult

Table 7.2: Robustness study results, sorted by exploration robustness, decreasing, then sorted alphabetically by Google Play category. 13 (50%) apps can be explored without problems; 5 (19%) require login; 2 (8%) get stuck; 5 (19%) terminate early and 1 (4%) cannot be launched. 2 (8%) problematic explorations are app's fault, not input generator; 2 (8%) are easy to fix; 1 (4%) is medium and 8 (30%) are difficult to fix.

Chapter 8

Conclusion

Sandbox mining, a method first introduced in this thesis, is conceptually simple, yet surprisingly powerful synthesis of system-level automatic test generation and sandboxing, especially when coupled with existing techniques, as explained in Section 3.4. Sandbox mining solves the problem stated in Section 2.2: It significantly improves security of existing programs while not degrading their usability, at little cost. Sandbox mining strength stems from the fact it is a white-listing approach to security, as opposed to traditional black-listing methods. We no longer try to find all possible vulnerabilities, risking possibly missing the most important ones. As Dijkstra stated:

> "Program testing can best show the presence of errors but never their absence."

In our context "errors" are malicious behaviors. Instead, we mine an explicit, human-readable behavior description of an application and confine any possible malicious behavior to policy codifying the mined behavior, forcing any malicious actions to "hide in plain sight".

The consequences of sandbox mining are far reaching. One can easily envision a future in which all programs are accompanied with their automatically mined, explicit, human-readable and enforceable behaviors. While such profiles would not be as comprehensible as manually written specifications or documentation, they still provide huge advantage over existing descriptions of program behaviors, which are mostly just nonexistent.

Overall, we believe the sandbox mining concept introduced in this thesis holds a promise of a future with much more understandable and secure pro-

grams.

8.1 Future work

While sandbox mining has big potential, this thesis is just the beginning. We introduced the concept, described its properties and did small, preliminary evaluation to show proof-of-concept implementation is indeed feasible on existing software and maintains in practice the theoretical claims of the method. We painted possible beneficial future in Section 3.6.

If we aim at making sandbox mining truly worldwide standard, we have to recognize that as sandbox mining has to work with existing software, it has to leverage test generators and sandboxes implemented for given platforms. Desktop, mobile, web, cloud, Internet of things. These are just broad categories, for which implementation details of sandbox mining components will vary significantly and pose very different technical challenges. Within given platform category there is still more variation. In mobile we have Android and iOS. In desktop we have Windows, macOS and Linux. For web we have multiple browsers, multiple front-end frameworks, back-end server frameworks and many cloud provider APIs. For IoT, situation is even more complicated, as ubiquitous standards have yet to emerge and the category encompasses humongous variety of devices, from a toaster to a self-driving car.

Assuming we have powerful enough test generators and secure enough sandboxes for given platform, now we will have to build infrastructure for comparing, sharing, updating and certifying sandboxes, serving the foundation of behavior specification and enforcement policy marketplaces envisioned in Section 3.6.

The immediate work however is in continuing the research into the concept on Android platform. The focus here is on further evaluation of the concept feasibility and its various extensions. Immediate aspect of sandbox mining that call for further research are:

Behavior saturation upper bound approximation At this point we don't have any means to determine all the possible behaviors given application can make. Knowing this would give us more confidence in the degree of behavior saturation observed during mining and give us insights how the test generators could be improved to speed up the saturation and make it more complete. One could use static analysis of resource files and bytecode to find out all the GUI elements given app has, as well as all the Android framework API method calls it potentially can make.

Robustness Our test generator DROIDMATE is essentially a two component distributed system, composed of the host machine and the Android device on which explorations are conducted. Every command issued to the device can potentially fail, and experience shows most of the commands indeed fail, if only rarely. At fault is either the Android device communication done with Android Debug Bridge, the UIAUTOMATOR framework or the explored app itself. To maintain the ability to conduct long explorations without human presence, test generators have to be capable of recovering from all possible failures. The problem is confounded by the fact the exploration strategy employed by the test generator often maintains model of the app GUI and current exploration state in the model. Various app behaviors, including crashes, have to be handled while retaining model state integrity.

Our test generator is already highly robust. It gracefully handles failures of all commands that have proven to be unstable so far, often retrying multiple times and attempting other recoveries before gracefully giving up and progressing to exploration of next app in the queue. Applying DROIDMATE to more diverse Android apps would undoubtedly uncover more not yet handled corner cases, e.g. yet unseen Android OS dialog boxes or new ways, in the context of GUI model state, that apps can crash. Increasing robustness would mean supporting graceful recovery from all these situations.

Android GUI model fidelity The GUI model of the currently employed exploration strategy only approximates the Android GUI model. For a faithful, high-fidelity representation of the Android GUI, the exploration strategy would have to accurately model both the Android activities lifecycle [63] as well as Android tasks and the back stack [67]. Doing this would significantly increase robustness, by decreasing errors caused by imprecise modelling.

Scalability DROIDMATE has now limited support for running explorations on multiple devices which requires some nontrivial manual setup. To conduct explorations on thousands of apps, we will have to automate the process of managing the explorations on multiple devices and aggregating the results.

Compatibility Ideally, we would like to show sandbox mining can work on all or almost all Android applications, no matter how complex. Android app bytecode needs to be modified to work with current implementation, and the app has to be resigned with different certificate. While this works

most of the time, some of the programs do remote certificate or bytecode integrity checks, quitting if the checks fail. By applying the monitoring approach adopted in Boxify we could solve this problem, as no app modifications would necessary any longer.

Replayability DROIDMATE currently logs the trace of its actions which can be used to reconstruct the exploration in detail. Ideally, however, DROIDMATE should generate UIAUTOMATOR-based tests which can be rerun stand-alone, without DROIDMATE.

Deterministic environment Even if we will have explorations codified as stand-alone tests, we still cannot control the environment, e.g. the responses returned from remote hosts or initial database contents. A challenge here is to also record all the environment contents made during original explorations and replay them in the same deterministic sequence.

Data fixtures Presence of appropriate data fixture is a major obstacle for explorations trying to cover more complex behaviors. Accessing some functionalities might require having appropriate database entries, files on file system, etc. Ability to automatically generate such fixtures is future work.

Initial configuration Special case of providing data fixtures is initial configuration of the apps, like e.g. login credentials. Many apps require account registration and subsequent login to it. Making test generator capable of automatically registering and logging is challenging. We could assume registration was done manually and once per app provide the login credentials as initial configuration, e.g. in a configuration file read by the test generator at exploration start. This however is still challenging, because test generator would need to know where to input these credentials.

Human-in-the-loop One solution to the problem of initial configuration described above is augmenting the mining process with manually provided interactions. The exploration could recognize that for given GUI screen or, more generally, app state, a manually prerecorded interaction sequence is available, and replay it. This way not only login problem would be solved, but also other, more challenging app states and GUI screens could be reached.

Exploration strategies Especially fertile ground for future work are exploration strategies. Current biased random approach works reasonably well

already. However, much more powerful strategies could be employed, leveraging methods like dynamic symbolic execution, genetic algorithms, finite state automatons, other metaheuristics or model-based approaches, to name a few.

Input modalities Right now our test generator interacts with apps GUI via UIAUTOMATOR as well as commands sent by ADB, like e.g. sending a message to start an app. However, other input modalities could be employed, like system-level broadcasts, data incoming from sensors, networking data, etc.

Enforcement policies In this work we focused on behavior enforcement policies that take into account API call signatures, possibly in context of GUI events. Yet, the policies could be much more fine-grained and advanced. Consider policies that distinguish between specific API call parameter values, current state of environment like contents of files on file system, sequence of API calls, currently running processes on the device or actions already conducted by the user.

Computed resources It is not always clear if given resource was already seen or not. Consider generated files with varying serial numbers embedded in their names. For example, photo app can add ID to file name of each photo taken. Most likely all files from given generated sequence should be treated as one, yet this might lead to increased attack surface, as attacker could pretend to just add next file in the sequence. This problem calls for further investigation.

Benchmarks Core benefit of sandbox mining is that usability of the sandboxed app doesn't suffer too much. To prove this, many actual user interactions with apps should be recorded and codified in repeatable deterministic tests, to serve as a ground truth used to determine how much given behavior enforcement policy degrades apps usability.

Glossary

When reading glossary entries please note that **bold text** denotes names of other terms present in the glossary. Only the first occurrences of such names in given entry description are emboldened.

AAPT An abbreviation of "Android Asset Packaging Tool", a tool from Android SDK used for various actions for building and inspecting **.apk** files.

ADB An abbreviation of "Application Debug Bridge", a tool from Android SDK used for various developer activities done from command line over an USB cable connected to an Android device.

API A shorthand for "Android framework API" [64].

API call A shorthand for "A call to **API method**", that is, "A call to a method being part of the Android framework API". During **exploration**, **event-bound API calls** are being **monitored**, which includes monitoring of API calls. In this thesis there is a notion of **distinct API calls**.

API method A shorthand for "A method being part of **API**", that is, "A method being part of the Android framework API".

API call saturation A type of **behavior saturation** where the **resource accesses** considered are **distinct API calls**.

See also **event-bound API call saturation**.

.apk An .apk file is an *Android package* file, i.e. a file containing an Android application. **ADB** can be used on an .apk file to install the contained Android app on an Android device.

Application being secured Same as **program being secured**.

AUE Abbreviation of "Application under exploration". AUE is an application on which the process of **exploration** is being conducted.

Behavior enforcement policy A set of rules pertaining to a program behavior enforced by a sandbox. During **sandboxing** the sandbox checks if a program behavior at runtime is allowed according to the policy. If it is not, a **sandbox violation** occurs and the behavior is **flagged**. A behavior enforcement policy is derived from **mined behavior**, the result of the **mining** phase of **sandbox mining**. Two behavior enforcement policies are proposed in this thesis: API call enforcement policy and event-bound API call enforcement policy. They are described in Chapter 5.

Behavior policy A shorthand for a **behavior enforcement policy**.

Behavior saturation As an **exploration** progresses, the count of distinct **resource accesses** observed is getting closer to maximum possible with given **exploration strategy**. The closer the count to the maximum, the bigger the behavior saturation. Two specific kinds of behavior saturation are considered in this thesis: **API call saturation** and **event-bound API call saturation**.

Distinct API call Two **API calls** are distinct if they are not in the same equivalence class as defined by relation described in Section 5.2.1.

Distinct event-bound API call Similar to **distinct API call**, but the equivalence relation is defined not only on **API calls**, but **event-bound API calls**. The relation is described in Subsection 5.3.1.

DSE An abbreviation of "Dynamic symbolic execution".

Enforcement policy A shorthand for a **behavior enforcement policy**.

Event-bound API call An **API call** paired with an event that triggered it, as described in Section 5.3. Event-bound API call can be viewed, and is equivalent to, an *(event, API call)* pair. Event-bound API calls are **monitored** during **exploration**. In this thesis there is a notion of **distinct event-bound API calls**.

Event-bound API call saturation A type of **behavior saturation** where the **resource accesses** considered are **distinct event-bound API calls**.

Exploration The process of applying inputs generated with an **input generator** on a **program being secured**, with the goal of **monitoring resource accesses** it makes during execution. Exploration is the first step of **mining** phase of **sandbox mining**. During exploration the input generator generates inputs according to **exploration strategy**. As a result of exploration an **exploration log** is obtained, which is processed further in the mining phase.

Exploration action A kind of input provided by an **input generator** that is an output of **exploration strategy** decision of what to do next as part of the process of **exploration**. In the implementation described in Chapter 4, an exploration action can be any of *click, long-click, press home, press back, reset, terminate*. In DROIDMATE implementation exploration actions are multi-step operations, executed with the help of UIAUTOMATOR framework and **ADB**. Some of the steps of the exploration actions are actions on the **GUI**, like *click*. Such actions are conducted via the UIAUTOMATOR framework and are called *GUI actions*.

Exploration log All the data obtained from an **exploration** in a form of a serialized Java object written out to the file system. Exploration logs are used by DROIDMATE to generate various human-readable data, like the textual **exploration summaries** and **behavior saturation** charts. When we mention in this thesis we had manually analyzed exploration logs, we, of course, first convert them to a human-readable format. Usually we leverage the standard DROIDMATE output, like exploration summaries, but at times we used custom-tailored one-off views of the data.

Exploration strategy The algorithm deciding which inputs should be generated with an **input generator** during an **exploration**. Exploration strategy might use information obtained from observing the explored program **GUI** and **resource accesses** it makes to determine which inputs should be generated next. Exploration strategies can be purely random, systematic, or other, as explained in Chapter 2. For this thesis a biased random exploration strategy was implemented, as described in Subsection 4.7.1.

Exploration summary Textual summary of an **exploration log**.

Explore To conduct **exploration**.

Flagging A sandbox flags a behavior if it violates a **behavior enforcement policy** during **sandboxing** phase. In the prototype implementation described in Chapter 5, as a result of flagging, the flagged behavior is either completely blocked or a pop-up box is displayed to the end-user with relevant, contextualized human-readable information, asking her to endorse or forbid the behavior. The user decision is remembered and applied automatically to future occurrences of equivalent behaviors. If the behavior is blocked, a predefined value is returned, like e.g. null.

GUI An abbreviation of "Graphical User Interface". Depending on the sentence context, GUI of a program may denote either all possible screens of given program, or just the currently visible screen of the program. A read-only representation of the currently visible screen is called a **GUI snapshot**.

GUI element A basic building block of a program **GUI**. In the context of this thesis this means basic building block of a GUI of an Android app, which is a **view**, also called **widget**.

GUI snapshot The read-only representation of the currently visible screen of a given program **GUI**. In the context of this thesis, GUI snapshot most often denotes the screen of an application running on an Android device, composed of **views**, represented by **XML window hierarchy**.

Host machine The part of DROIDMATE that runs on the developer/user machine, as opposed to the DROIDMATE code running on the device. The code running on the device is either the UIAUTOMATOR-daemon or the **inlined monitor** within the **AUE**.

IDS An abbreviation of "Intrusion detection system".

Inlining The process of rewriting the bytecode of the **.apk** file of Android app to add **monitor** to it. The monitor enables monitoring of the **API calls** made by the app during an **exploration**.

Input generator Input generator is a tool that can automatically generate inputs for a given program and execute the program with those inputs. An input generator is capable of conducting **exploration** if it can (a) **monitor resource accesses** and (b) determine which inputs to generate based on the program and/or system environment status obtained after applying previously generated inputs. In such case, the inputs generated and the order in which they are fed to the program is determined by **exploration**

strategy. For example, an input generator can explore by analyzing the **GUI** of a Java program, finding clickable elements and clicking them. In this thesis, the name *input generator* is used interchangeably with **test generator**.

Mined behavior A set of **resource accesses** that are extracted from **exploration log**, which in turn is obtained from **exploration**. In this thesis the mined behavior consists of **event-bound API calls**. **Behavior enforcement policies** are derived from the mined behavior.

Mined policy In the context of this thesis mined policy is a **behavior enforcement policy**. Mined policy is a shortcut for saying "a behavior enforcement policy inferred from **mined behavior**, extracted from **exploration summary**, obtained from **exploration**, done in the **mining phase** of **sandbox mining**".

Mining, mining phase The first phase of the **sandbox mining concept**. In this phase first an **exploration** of **AUE** is conducted, resulting in an **exploration log**. Next, **mined behavior** is extracted from the exploration log. The mined behavior serves as a basis of **behavior enforcement policy** used in the **sandboxing** phase of sandbox mining.

Monitor, monitoring Monitor is a software component that enables monitoring of **resource accesses** made during **exploration**. Monitoring is the process of intercepting resource accesses made during exploration, with the goal of persisting them in an **exploration log**. One can also say monitoring is the process of *recording* resource accesses.

Policy In the context of this thesis policy is a **behavior enforcement policy**.

Program being secured A program to which the concrete implementation of the **sandbox mining concept** is being applied. In practice this means that program being secured is also an **AUE**.

Resource A part of system environment which given program may access at runtime, resulting in a **resource access**. Examples of resources include: OS API methods, network connections, files on a file system, handles to processes, databases, environment variables. In the implementation done as part of this thesis, the resource accesses considered are **event-bound API calls**.

Resource access Resource access happens when a program interacts with a **resource**. The specific way of interaction depends on the type of resource. OS API methods are accessed by calling them. Files on a file system are accessed by opening them, reading their contents, deleting them, etc. Network connections are accessed by sending or receiving data from a remote computer or similar.

As an example, if a program calls method `File.open("foo.txt")` from an OS API, two resource accesses have been made. One to the resource *file foo.txt*, as it has been opened, and one to the resource *method File.open()*, as the method has been called.

Resource accesses are **monitored** during **exploration**. In this thesis the monitored resource accesses are **event-bound API calls**.

Sandbox mining, sandbox mining concept The idea introduced by this thesis, composed of two stages: **mining** and **sandboxing**. For details, see Chapter 3.

Sandbox policy, sandbox enforcement policy In the context of this thesis sandbox (enforcement) policy is a **behavior enforcement policy**.

Sandbox violation Happens when sandbox has to **flag** a behavior because it is not allowed by currently enforced **behavior enforcement policy**.

Sandboxing, sandboxing phase Second phase of the **sandbox mining** concept. Sandboxing happens when running in production the **program being secured**. The program is run in a sandbox that enforces a **behavior enforcement policy** that was derived from **mined behavior** obtained from the **mining** phase of sandbox mining.

Saturation In the context of this thesis saturation means **behavior saturation**.

Security policy In the context of this thesis security policy is a **behavior enforcement policy**.

Test generator Another name for **input generator**. To be precise, the inputs generated with input generator can be either executed directly on the subject program, or can be codified in automated tests which are then executed, applying the inputs to the subject program. The test generator has the ability to codify the inputs in tests, while input generator not

necessarily. For this thesis this distinction is irrelevant and so the names *test generator* and *input generator* are used interchangeably.

View In context of a **GUI** of Android app, a view is a the basic building block of the GUI and is represented by the `View` class [70]. The XML representation of views, including their attributes, can be read from **XML window hierarchy**. In this thesis we use the terms *view*, **widget** and **GUI element** interchangeably, even though strictly speaking *widgets* are an interactive subtype of views [70].

Widget A subtype of **view** that is interactive [70]. A simple example of a widget type is the `Button` class [66]. In this thesis the term *widget* is used interchangeably with the term **GUI element** and the term *view*.

XML window hierarchy The XML window hierarchy [69] is a representation of Android **GUI snapshot**, composed of **views**. The window hierarchy can be obtained by making a call to appropriate method exposed by the API of UIAUTOMATOR [68].

Bibliography

[1] Adamsen, C. Q., Mezzetti, G., and Møller, A. Systematic execution of Android test suites in adverse conditions. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (2015), ISSTA 2015, ACM, pp. 83–93.

[2] Amalfitano, D., Fasolino, A. R., Tramontana, P., De Carmine, S., and Memon, A. M. Using GUI ripping for automated testing of Android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (2012), ASE 2012, ACM, pp. 258–261.

[3] Amalfitano, D., Fasolino, A. R., Tramontana, P., Ta, B. D., and Memon, A. M. MobiGUITAR: Automated model-based testing of mobile apps. *IEEE Software 32*, 5 (Sept 2015), 53–59.

[4] Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., McMinn, P., Bertolino, A., Li, J. J., and Zhu, H. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software 86*, 8 (2013), 1978 – 2001.

[5] Anand, S., Naik, M., Harrold, M. J., and Yang, H. Automated concolic testing of smartphone apps. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (2012), FSE '12, ACM, pp. 59:1–59:11.

[6] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware

taint analysis for Android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2014), PLDI '14, ACM, pp. 259–269.

[7] Au, K. W. Y., Zhou, Y. F., Huang, Z., and Lie, D. PScout: Analyzing the Android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (2012), CCS '12, ACM, pp. 217–228.

[8] Azim, T., and Neamtiu, I. Targeted and depth-first exploration for systematic testing of Android apps. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications* (2013), OOPSLA '13, ACM, pp. 641–660.

[9] Backes, M., Bugiel, S., Hammer, C., Schranz, O., and von Styp-Rekowsky, P. Boxify: Full-fledged app sandboxing for stock Android. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.* (2015), pp. 691–706.

[10] Backes, M., Gerling, S., Hammer, C., Maffei, M., and von Styp-Rekowsky, P. AppGuard–fine-grained policy enforcement for untrusted Android applications. In *Data Privacy Management and Autonomous Spontaneous Security*, J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boulahia, S. Foley, and W. M. Fitzgerald, Eds., Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 213–231.

[11] Ball, T., and Rajamani, S. K. The SLAM project: Debugging system software via static analysis. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (2002), POPL '02, ACM, pp. 1–3.

[12] Bhoraskar, R., Han, S., Jeon, J., Azim, T., Chen, S., Jung, J., Nath, S., Wang, R., and Wetherall, D. Brahmastra: Driving apps to test the security of third-party components. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.* (2014), pp. 1021–1036.

[13] Bierma, M., Gustafson, E., Erickson, J., Fritz, D., and Choe, Y. R. Andlantis: Large-scale Android dynamic analysis. *CoRR abs/1410.7751* (2014).

[14] Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S., and Albayrak, S. An Android application sandbox system for suspicious software detection. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on* (Oct 2010), pp. 55–62.

[15] Böhme, M., and Soumya, P. On the efficiency of automated testing. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014), FSE 2014, ACM, pp. 632–642.

[16] Boyer, R. S., Elspas, B., and Levitt, K. N. Select—a formal system for testing and debugging programs by symbolic execution. In *Proceedings of the International Conference on Reliable Software* (1975), ACM, pp. 234–245.

[17] Carter, P., Mulliner, C., Lindorfer, M., Robertson, W., and Kirda, E. CuriousDroid: automated user interface interaction for android application analysis sandboxes. In *International Conference on Financial Cryptography and Data Security* (2016), Springer, pp. 231–249.

[18] Choi, W., Necula, G., and Sen, K. Guided GUI testing of Android apps with minimal restart and approximate learning. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications* (2013), OOPSLA '13, ACM, pp. 623–640.

[19] Choudhary, S. R., Gorla, A., and Orso, A. Automated test input generation for Android: Are we there yet? *CoRR abs/1503.07217* (2015).

[20] Dallmeier, V., Burger, M., Orth, T., and Zeller, A. WebMate: Generating test cases for web 2.0. In *Software Quality. Increasing Value in Software and Systems Development* (Jan. 2013), Springer, pp. 55–69.

[21] Denning, D. E. An intrusion-detection model. *IEEE Trans. Softw. Eng. 13*, 2 (Feb. 1987), 222–232.

[22] Ernst, M. D. Static and dynamic analysis: Synergy and duality. In *WODA 2003: ICSE Workshop on Dynamic Analysis* (2003), pp. 24–27.

[23] Ernst, M. D. Static and dynamic analysis: synergy and duality. In *Proc. PASTE '04* (2004), ACM, pp. 35–35.

[24] Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (2011), CCS '11, ACM, pp. 627–638.

[25] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (2012), SOUPS '12, ACM, pp. 3:1–3:14.

[26] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (1996), SP '96, IEEE Computer Society, pp. 120–.

[27] Fraser, G., and Arcuri, A. Evolutionary generation of whole test suites. In *International Conference On Quality Software (QSIC)* (2011), IEEE Computer Society, pp. 31–40.

[28] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security 28*, 1 (2009), 18–28.

[29] Godefroid, P., Klarlund, N., and Sen, K. Dart: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (2005), PLDI '05, ACM, pp. 213–223.

[30] Godefroid, P., Levin, M. Y., and Molnar, D. Automated whitebox fuzz testing. In *Proceedings of Network and Distributed Systems Security (NDSS 2008)* (July 2008), pp. 151–166.

[31] Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (2014), ICSE 2014, ACM, pp. 1025–1035.

[32] Gross, F., Fraser, G., and Zeller, A. Exsyst: Search-based gui testing (tool paper). In *34th International Conference on Software Engineering* (2012), IEEE, pp. 1423–1426.

[33] Hamlet, R. Random testing. In *Encyclopedia of Software Engineering* (1994), Wiley, pp. 970–978.

[34] Hao, S., Liu, B., Nath, S., Halfond, W. G., and Govindan, R. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (2014), MobiSys '14, ACM, pp. 204–217.

[35] Holler, C., Herzig, K., and Zeller, A. Fuzzing with code fragments. In *Proceedings of the 21st USENIX Conference on Security Symposium* (2012), Security'12, USENIX Association, pp. 38–38.

[36] Holzmann, G. J. The model checker SPIN. *IEEE Trans. Softw. Eng. 23*, 5 (May 1997), 279–295.

[37] Patent application EP3259697 - MINING SANDBOXES. https://register.epo.org/application?number=EP16709293, Retrieved March 2018.

[38] Jamrozik, K., von Styp-Rekowsky, P., and Zeller, A. Mining sandboxes. In *Proceedings of the 38th International Conference on Software Engineering* (2016), ICSE '16, ACM, pp. 37–48.

[39] Jamrozik, K., and Zeller, A. DroidMate: A robust and extensible test generator for Android. In *Proceedings of the International Conference on Mobile Software Engineering and Systems* (2016), MOBILESoft '16, ACM, pp. 293–294.

[40] Kruegel, C., Vigna, G., and Robertson, W. A multi-model approach to the detection of web-based attacks. *Computer Networks 48*, 5 (2005), 717–738.

[41] Langner, R. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy 9*, 3 (2011), 49–51.

[42] Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., and Kirda, E. AccessMiner: Using system-centric models for malware protection. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (2010), CCS '10, ACM, pp. 399–412.

[43] Li, Y., Yang, Z., Guo, Y., and Chen, X. DroidBot: a lightweight UI-Guided test input generator for Android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (May 2017), pp. 23–26.

[44] Linares-Vásquez, M., White, M., Bernal-Cárdenas, C., Moran, K., and Poshyvanyk, D. Mining Android app usages for generating actionable GUI-based execution scenarios. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (2015), MSR '15, IEEE Press, pp. 111–122.

[45] Lu, L., Li, Z., Wu, Z., Lee, W., and Jiang, G. CHEX: Statically vetting Android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (2012), CCS '12, ACM, pp. 229–240.

[46] Machiry, A., Tahiliani, R., and Naik, M. Dynodroid: An input generation system for Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (2013), ESEC/FSE 2013, ACM, pp. 224–234.

[47] Mahmood, R., Mirzaei, N., and Malek, S. EvoDroid: Segmented evolutionary testing of Android apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014), FSE 2014, ACM, pp. 599–609.

[48] Mao, K., Harman, M., and Jia, Y. Sapienz: Multi-objective automated testing for Android applications. In *Proc. of ISSTA'16* (2016), pp. 94–105.

[49] Mesbah, A., van Deursen, A., and Lenselink, S. Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB) 6*, 1 (2012), 3:1–3:30.

[50] Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., Vendome, C., and Poshyvanyk, D. Automatically discovering, reporting and reproducing Android application crashes. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)* (April 2016), pp. 33–44.

[51] Nielson, F., Nielson, H. R., and Hankin, C. *Principles of Program Analysis*. Springer-Verlag New York, Inc., 1999.

[52] Pacheco, C., Lahiri, S. K., Ernst, M. D., and Ball, T. Feedback-directed random test generation. In *Proceedings of the 29th International Conference on Software Engineering* (2007), ICSE '07, IEEE Computer Society, pp. 75–84.

[53] Rastogi, V., Chen, Y., and Enck, W. AppsPlayground: Automatic security analysis of smartphone applications. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy* (2013), CODASPY '13, ACM, pp. 209–220.

[54] Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H. J., and Cowan, C. User-driven access control: Rethinking permission granting in modern operating systems. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (2012), SP '12, IEEE Computer Society, pp. 224–238.

[55] Saltzer, J., and Schroeder, M. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (Sept 1975), 1278–1308.

[56] Shirley, J., and Evans, D. The user is not the enemy: Fighting malware by tracking user intentions. In *Proceedings of the 2008 Workshop on New Security Paradigms* (2008), NSPW '08, ACM, pp. 33–45.

[57] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (2010), SP '10, IEEE Computer Society, pp. 305–316.

[58] Tillmann, N., and de Halleux, N. J. Pex — white box test generation for .NET. In *Proc. TAP* (2008), pp. 134–153.

[59] van der Merwe, H., van der Merwe, B., and Visser, W. Verifying Android applications using Java PathFinder. *SIGSOFT Softw. Eng. Notes 37*, 6 (Nov. 2012), 1–5.

[60] Wong, M. Y., and Lie, D. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. In *NDSS* (2016).

[61] Yang, W., Prasad, M. R., and Xie, T. A grey-box approach for automated GUI-model generation of mobile applications. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering* (2013), FASE'13, Springer-Verlag, pp. 250–265.

[62] Android 6 permission model. https://developer.android.com/training/articles/user-data-permissions.html#version_specific_details_permissions_in_m. Retrieved May 2017.

[63] Android lifecycle. https://developer.android.com/guide/components/activities/activity-lifecycle.html. Retrieved May 2017.

[64] Android API reference. https://developer.android.com/reference/packages.html. Retrieved May 2017.

[65] Android AppManifest. https://developer.android.com/guide/topics/manifest/manifest-intro.html. Retrieved May 2017.

[66] Android Button class. https://developer.android.com/reference/android/widget/Button.html. Retrieved May 2017.

[67] Android tasks and back stack. https://developer.android.com/guide/components/tasks-and-back-stack.html. Retrieved May 2017.

[68] Android UI Automator API dumpWindowHierarchy method. https://developer.android.com/reference/android/support/test/uiautomator/UiDevice.html#dumpWindowHierarchy(java.io.File). Retrieved May 2017.

[69] Android UI overview. https://developer.android.com/guide/topics/ui/overview.html#Layout. Retrieved May 2017.

[70] Android View class. https://developer.android.com/reference/android/view/View.html. Retrieved May 2017.

[71] AndroTest website. http://bear.cc.gatech.edu/~shauvik/androtest/. Retrieved May 2017.

[72] ArtHook source. https://github.com/mar-v-in/ArtHook. Retrieved May 2017.

[73] Java method declaration spec. https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.4. Retrieved May 2017.

[74] PUMA source. https://github.com/USC-NSL/PUMA. Retrieved May 2017.

[75] Android system permissions: mirror of previous documentation. https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/security/permissions.html. Retrieved May 2017.

[76] Hackers remotely kill a Jeep on the highway—with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. Retrieved May 2017.

[77] Linkedin hack. https://en.wikipedia.org/wiki/2012_LinkedIn_hack. Retrieved May 2017.

[78] Adobe hack. https://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users. Retrieved May 2017.

[79] Tumblr hack. https://www.theguardian.com/technology/2016/may/31/tumblr-emails-for-sale-darknet-65-million-hack-passwords. Retrieved May 2017.

[80] CVEs (Common Vulnerabilities and Exposures) list. https://www.cvedetails.com/. Retrieved May 2017.

[81] Heartbleed. http://heartbleed.com/. Retrieved May 2017.

[82] Windows user account control. https://en.wikipedia.org/wiki/User_Account_Control. Retrieved May 2017.

[83] Android permission groups. https://developer.android.com/guide/topics/permissions/requesting.html#perm-groups. Retrieved May 2017.

[84] Using Android permissions. https://developer.android.com/guide/topics/permissions/requesting.html#permissions. Retrieved May 2017.

[85] Android security architecture: mirror of previous documentation. https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/security/permissions.html#arch. Retrieved May 2017.

[86] Android application security. https://source.android.com/security/overview/app-security. Retrieved May 2017.

[87] Android, normal permissions. https://developer.android.com/guide/topics/permissions/normal-permissions.html. Retrieved May 2017.

[88] Android normal vs dangerous permissions. https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous. Retrieved May 2017.

[89] Android security. https://source.android.com/security/. Retrieved May 2017.

[90] Inernet permission. https://developer.android.com/reference/android/Manifest.permission.html#INTERNET. Retrieved May 2017.

[91] Android LocationManager. https://developer.android.com/reference/android/location/LocationManager.html. Retrieved May 2017.

[92] Metamorphic code. https://en.wikipedia.org/wiki/Metamorphic_code. Retrieved May 2017.

[93] Firewall. https://en.wikipedia.org/wiki/Firewall_(computing). Retrieved May 2017.

[94] Deep packet inspection. https://en.wikipedia.org/wiki/Deep_packet_inspection. Retrieved May 2017.

[95] Intrusion detection system. https://en.wikipedia.org/wiki/Intrusion_detection_system. Retrieved May 2017.

[96] Evasive malware report. http://labs.lastline.com/evasive-malware-gone-mainstream. Retrieved May 2017.

[97] Android framework. https://developer.android.com/guide/platform/index.html#api-framework. Retrieved May 2017.

[98] Android SDK initial release date. https://en.wikipedia.org/wiki/Android_software_development#SDK. Retrieved May 2017.

[99] Android ProGuard. https://developer.android.com/studio/build/shrink-code.html. Retrieved May 2017.

[100] Android Monkey: UI/Application Exerciser. https://developer.android.com/studio/test/monkey.html. Retrieved May 2017.

[101] UI Automator: Testing UI for multiple apps. https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html. Retrieved May 2017.

[102] DroidMate webpage (redirects to BoxMate webpage). http://www.droidmate.org. Retrieved May 2017.

[103] GitHub repository of DroidMate. https://github.com/konrad-jamrozik/droidmate. Retrieved May 2017.

[104] Google Play Store. https://play.google.com/. Retrieved May 2017.

[105] A3E website. http://spruce.cs.ucr.edu/a3e/. Retrieved May 2017.

[106] SwiftHand source. https://github.com/wtchoi/SwiftHand/. Retrieved May 2017.

[107] Dynodroid source. https://github.com/Machiry/ddator. Retrieved May 2017.

[108] Acteve source. https://github.com/saswatanand/acteve/. Retrieved May 2017.

[109] IntelliDroid source. https://github.com/miwong/IntelliDroid. Retrieved March 2018.

[110] CuriousDroid source. https://github.com/pdcarter/curiousdroid/tree/master/edu/neu/ccs/curiousdroid. Retrieved March 2018.

[111] DroidBot source. https://github.com/honeynet/droidbot. Retrieved March 2018.

Appendices

Appendix A

Using and extending DroidMate

DROIDMATE is free software [103], licensed under GPL-3.0. It boasts continuous integration server [103] and extensive documentation [103], explaining how to build it, run, test, deploy, develop, extend and troubleshoot. It includes programmatic API usage examples and comprehensive suite of tests run during the build, as well as additional Android device integration tests users can run to validate their DROIDMATE installation. Please refer to the documentation [103] to see how to use and extend DROIDMATE.

Appendix B

Monitored Android API methods list

This appendix contains the list of sensitive Android 4.4.2 (API 19) method signatures of Android framework API which have been monitored in the experiments described in Chapter 6. There are 97 methods in total. The format used is a fully qualified method signature formatted according to [73]. In case of long method signatures, first the common prefix is listed, followed by a colon.

```
android.app.ActivityManager.getRecentTasks(int, int)
android.app.ActivityManager.getRunningTasks(int)
android.bluetooth.BluetoothHeadset.startVoiceRecognition(android.bluetooth.BluetoothDevice)
android.bluetooth.BluetoothHeadset.stopVoiceRecognition(android.bluetooth.BluetoothDevice)

android.content.ContentProviderClient:
.bulkInsert(android.net.Uri, android.content.ContentValues[])
.delete(android.net.Uri, java.lang.String, java.lang.String[])
.insert(android.net.Uri, android.content.ContentValues)
.openFile(android.net.Uri, java.lang.String)
.query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)
.update(android.net.Uri, android.content.ContentValues, java.lang.String, java.lang.String[])

android.content.ContentResolver:
.bulkInsert(android.net.Uri, android.content.ContentValues[])
.delete(android.net.Uri, java.lang.String, java.lang.String[])
.insert(android.net.Uri, android.content.ContentValues)
.openInputStream(android.net.Uri)
.registerContentObserver(android.net.Uri, boolean, android.database.ContentObserver)
.update(android.net.Uri, android.content.ContentValues, java.lang.String, java.lang.String[])

android.hardware.Camera.open()
android.hardware.Camera.open(int)

android.location.LocationManager:
.addProximityAlert(double, double, float, long, android.app.PendingIntent)
.addTestProvider(java.lang.String, boolean, boolean, boolean, boolean, boolean, boolean, boolean, int, int)
.clearTestProviderEnabled(java.lang.String)
.clearTestProviderLocation(java.lang.String)
.clearTestProviderStatus(java.lang.String)
.getBestProvider(android.location.Criteria, boolean)
.getLastKnownLocation(java.lang.String)
```

```
.getProvider(java.lang.String)
.getProviders(android.location.Criteria, boolean)
.getProviders(boolean)
.isProviderEnabled(java.lang.String)
.removeTestProvider(java.lang.String)
.requestLocationUpdates(long, float, android.location.Criteria, android.app.PendingIntent)
.requestLocationUpdates(long, float, android.location.Criteria, android.location.LocationListener, android.os.Looper)
.requestLocationUpdates(java.lang.String, long, float, android.app.PendingIntent)
.requestLocationUpdates(java.lang.String, long, float, android.location.LocationListener)
.requestLocationUpdates(java.lang.String, long, float, android.location.LocationListener, android.os.Looper)
.requestSingleUpdate(android.location.Criteria, android.app.PendingIntent)
.requestSingleUpdate(android.location.Criteria, android.location.LocationListener, android.os.Looper)
.requestSingleUpdate(java.lang.String, android.app.PendingIntent)
.requestSingleUpdate(java.lang.String, android.location.LocationListener, android.os.Looper)
.sendExtraCommand(java.lang.String, java.lang.String, android.os.Bundle)
.setTestProviderEnabled(java.lang.String, boolean)
.setTestProviderLocation(java.lang.String, android.location.Location)
.setTestProviderStatus(java.lang.String, int, android.os.Bundle, long)

android.media.AudioManager.isBluetoothA2dpOn()
android.media.AudioManager.isWiredHeadsetOn()
android.media.AudioManager.setBluetoothScoOn(boolean)
android.media.AudioManager.setMicrophoneMute(boolean)
android.media.AudioManager.setMode(int)
android.media.AudioManager.setParameter(java.lang.String, java.lang.String)
android.media.AudioManager.setParameters(java.lang.String)
android.media.AudioManager.setSpeakerphoneOn(boolean)
android.media.AudioManager.startBluetoothSco()
android.media.AudioManager.stopBluetoothSco()
android.media.AudioRecord.<init>(int, int, int, int, int)
android.media.MediaPlayer.setWakeMode(android.content.Context, int)
android.media.MediaRecorder.setAudioSource(int)
android.media.MediaRecorder.setVideoSource(int)
android.net.ConnectivityManager.requestRouteToHost(int, int)
android.net.ConnectivityManager.setNetworkPreference(int)
android.net.ConnectivityManager.startUsingNetworkFeature(int, java.lang.String)
android.net.ConnectivityManager.stopUsingNetworkFeature(int, java.lang.String)
android.net.wifi.WifiManager$MulticastLock.acquire()
android.net.wifi.WifiManager$MulticastLock.release()
android.net.wifi.WifiManager$WifiLock.acquire()
android.net.wifi.WifiManager$WifiLock.release()
android.net.wifi.WifiManager.addNetwork(android.net.wifi.WifiConfiguration)
android.net.wifi.WifiManager.disableNetwork(int)
android.net.wifi.WifiManager.disconnect()
android.net.wifi.WifiManager.enableNetwork(int, boolean)
android.net.wifi.WifiManager.pingSupplicant()
android.net.wifi.WifiManager.reassociate()
android.net.wifi.WifiManager.reconnect()
android.net.wifi.WifiManager.removeNetwork(int)
android.net.wifi.WifiManager.saveConfiguration()
android.net.wifi.WifiManager.setWifiEnabled(boolean)
android.net.wifi.WifiManager.startScan()
android.os.PowerManager$WakeLock.acquire()
android.os.PowerManager$WakeLock.acquire(long)
android.os.PowerManager$WakeLock.release(int)
android.speech.SpeechRecognizer.cancel()
android.speech.SpeechRecognizer.startListening(android.content.Intent)
android.speech.SpeechRecognizer.stopListening()
android.telephony.TelephonyManager.getCellLocation()
android.telephony.TelephonyManager.getDeviceId()
android.telephony.TelephonyManager.getDeviceSoftwareVersion()
android.telephony.TelephonyManager.getLinelNumber()
android.telephony.TelephonyManager.getNeighboringCellInfo()
android.telephony.TelephonyManager.getSimSerialNumber()
android.telephony.TelephonyManager.getSubscriberId()
android.telephony.TelephonyManager.getVoiceMailAlphaTag()
android.telephony.TelephonyManager.getVoiceMailNumber()
android.telephony.TelephonyManager.listen(android.telephony.PhoneStateListener, int)
java.net.Socket.<init>(java.lang.String, int, java.net.InetAddress, int)
java.net.Socket.<init>(java.lang.String, int, boolean)
java.net.Socket.<init>(java.net.InetAddress, int, java.net.InetAddress, int)
java.net.Socket.<init>(java.net.InetAddress, int, boolean)
java.net.URL.openConnection()
```

Appendix C

Exploration summaries

This appendix contains DROIDMATE exploration summaries of the 2 hour explorations of twelve apps from Chapter 6 evaluation set, plus 3.5 hour long exploration of SNAPCHAT 4.1.07. While reading the summaries, take note that:

- Each summary is organized by app and exploration identifiers, with the identifiers surrounded by line separators of ===.
- For each app, we first list the resource accesses observed while replaying the automated tests representing real world use cases. Such entries have "use-case" in their titles, for example,
  `use-case:viewDocument:com.adobe.reader`.

  First we give hand-written description of the use case. Next, after a line separator of ---, we list observed API calls, then, after another --- line separator, we list observed *(event, API call)* pairs.
- The listing of resource accesses made during execution of use cases shows in the column "DroidMate" at which time the same API call or *(event, API call)* pair was observed during exploration made with inputs automatically generated with DROIDMATE. If it wasn't seen by DROIDMATE, it says "None!".
- Each API call is listed as "TId" (Thread Identifier), signature, and possible argument information, such as an URI, if relevant for distinguishing API calls.

- After the use cases comes the === line separator and the DROIDMATE exploration summary of the same app is listed. Again, first API calls, then *(event, API call)* pairs.

```
================================================================
use-case:addAndEditExpense:at.markushi.expensemanager
================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the plus sign in the upper right corner to record a new expense.
//  3. Set the first digit of the expense, '1'.
//  4. Set the second digit of the expense, '5'.
//  5. Press the decimal sign while setting expense value.
//  6. Set the third digit of the expense, '8'. The expense value is now 15.80.
//  7. Confirm the expense value.
//  8. Save the new expense.
//  9. Unroll the expense category in the overview, to display the expense itself.
// 10. Click the expense in the overview.
// 11. Set expense category to 'Health'.
// 12. Click 'edit note'.
// 13. Set the note text to 'Pills'.
// 14. Save the modified expense.
// 15. Terminate the exploration.

Total run time:         0m 34s
Total actions count:   15 (including the final action terminating exploration)

----------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature

----------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                          API signature


================================================================
use-case:deleteExpenseFromHistory:at.markushi.expensemanager
================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Open overview menu.
//  3. Click the 'history' tab.
//  4. Click the only entry.
//  5. Delete the entry.
```

```
//  6. Terminate the exploration.

Total run time:        10m  7s
Total actions count:    6 (including the final action terminating exploration)

------------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature

------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                    API signature


==================================================================
use-case:viewAndSetBudget:at.markushi.expensemanager
==================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Open the overview menu.
//  3. Click the 'budget' tab.
//  4. Click the 'Other' category.
//  5. Click the '7' digit.
//  6. Click the 'OK' button to confirm the budget for the 'Other' category.
//  7. Terminate the exploration.

Total run time:         0m 15s
Total actions count:    7 (including the final action terminating exploration)

------------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature

------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
```

```
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                    API signature


============================================================
DroidMate-run:at.markushi.expensemanager
============================================================

Total run time:       120m 19s
Total actions count: 2558 (including the final action terminating exploration)
Total resets count:   107 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                    API signature


============================================================
use-case:viewDocument:com.adobe.reader
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'What's New' button.
//  3. Click the 'Help' button to open side menu.
//  4. Click the 'Documents' tab in the side menu.
//  5. Click the first (topmost) document to open it.
//  6. Terminate the exploration.

Total run time:         0m 17s
Total actions count:    6 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
```

```
------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                    API signature


============================================================
DroidMate-run:com.adobe.reader
============================================================

Total run time:       120m 19s
Total actions count: 2576 (including the final action terminating exploration)
Total resets count:    96 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 1

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   8m 57s  191 TId:  702 java.net.Socket: void <init>

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 1

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                  API signature
   8m 57s  191 background                                           TId:  702 java.net.Socket: void <init>


============================================================
use-case:scan:com.antivirus
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'Activate' button.
//  3. Click the 'Scan Now' button.
//  4. Wait until the 'View Scan Results' button appears and click it.
//  5. Terminate the exploration.

Total run time:        12m 42s
Total actions count:    5 (including the final action terminating exploration)

------------------------------------------------------------
```

```
Unique API calls count observed in the run: 13

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  4s    1 |    0m  4s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://sms/
   0m  4s    1 |    0m  4s    1 TId:    1 android.telephony.TelephonyManager: java.lang.String getLinelNumber()
   0m  4s    1 |    0m  4s    1 TId:    1 android.telephony.TelephonyManager: java.lang.String getSimSerialNumber()
   0m  4s    1 |    0m  4s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://settings/system
   0m  4s    1 |    0m  4s    1 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://call_log/calls
   0m 13s    3 |    0m 14s    3 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://com.android.chrome.browser/history
   0m 13s    3 |    0m 14s    3 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 13s    3 |    0m 14s    3 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://browser/searches
   0m 13s    3 |    0m 14s    3 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://browser/bookmarks
   0m 13s    3 |    0m 14s    3 TId: 1159 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://com.android.browser/history
   0m 13s    3 |    0m 14s    3 TId: 1156 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   0m 14s    3 |    0m 15s    3 TId: 1159 java.net.URL: java.net.URLConnection openConnection()
   0m 13s    3 |    0m 15s    3 TId: 1159 java.net.Socket: void <init>

------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 16

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m  4s    1 |    0m  4s    1 <reset>                                                               TId:    1 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://sms/
   0m  4s    1 |    0m  4s    1 <reset>                                                               TId:    1 android.telephony.TelephonyManager: java.lang.String
          getLinelNumber()
   0m  4s    1 |    0m  4s    1 <reset>                                                               TId:    1 android.telephony.TelephonyManager: java.lang.String
          getSimSerialNumber()
   0m  4s    1 |    0m  4s    1 <reset>                                                               TId:    1 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://settings/system
   0m  4s    1 |    0m  4s    1 <reset>                                                               TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://call_log/calls
   0m 13s    3 |    0m 14s    3 background                                                            TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/history
   0m 13s    3 |    0m 14s    3 background                                                            TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 13s    3 |    0m 14s    3 background                                                            TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://browser/searches
   0m 13s    3 |    0m 14s    3 background                                                            TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://browser/bookmarks
```

```
   0m 13s    3 |    0m 14s    3 background                                                    TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.browser/history
   0m 13s    3 |    0m 14s    3 background                                                    TId: 1156 android.telephony.TelephonyManager: java.lang.String
          getSimSerialNumber()
   0m 13s    3 |    0m 14s    3 background                                                    TId: 1156 android.telephony.TelephonyManager: java.lang.String
          getDeviceId()
   0m 13s    3 |    0m 14s    3 background                                                    TId: 1159 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://sms/
   0m 14s    3 |    0m 15s    3 background                                                    TId: 1159 android.telephony.TelephonyManager: java.lang.String
          getLinelNumber()
   0m 14s    3 |    0m 15s    3 background                                                    TId: 1159 java.net.URL: java.net.URLConnection openConnection
         ()
   0m 13s    3 |    0m 15s    3 background                                                    TId: 1159 java.net.Socket: void <init>


============================================================
DroidMate-run:com.antivirus
============================================================

Total run time:       120m 30s
Total actions count: 1606 (including the final action terminating exploration)
Total resets count:   114 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 54

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  4s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://sms/
   0m  4s    1 TId:    1 android.telephony.TelephonyManager: java.lang.String getLinelNumber()
   0m  4s    1 TId:    1 android.telephony.TelephonyManager: java.lang.String getSimSerialNumber()
   0m  4s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://settings/system
   0m  4s    1 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://call_log/calls
   0m 13s    3 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://com.android.chrome.browser/history
   0m 13s    3 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://com.android.chrome.browser/bookmarks
   0m 13s    3 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://browser/searches
   0m 13s    3 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://browser/bookmarks
   0m 13s    3 TId:  941 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://com.android.browser/history
   0m 13s    3 TId:  938 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   0m 14s    4 TId:  941 java.net.URL: java.net.URLConnection openConnection()
   0m 15s    4 TId:  941 java.net.Socket: void <init>
  17m 49s  211 TId: 1022 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
        String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.chrome.browser/bookmarks
  28m 37s  342 TId: 1092 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
        browser/bookmarks
  28m 43s  343 TId: 1093 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
```

```
      browser/history
28m 43s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://sms
28m 44s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://logs/historys
28m 44s  343 TId: 1094 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
      String[],java.lang.String,android.os.CancellationSignal) uri: content://logs/historys
28m 44s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://sms/inbox
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://sms/sent
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms/inbox
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms-sms
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms/address
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms/part
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms/sent
28m 45s  343 TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://mms/outbox
28m 45s  343 TId: 1095 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://call_log/calls
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/contacts
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/data
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/raw_contact_entities
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/emails
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/postals
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/groups
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
      contacts/groups_summary
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/people
28m 45s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/phones
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/photos
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/calls
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/settings
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      deleted_people
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/people/
      with_email_or_im_filter
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/groups
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      deleted_groups
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      groupmembership
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      groupmembershipraw
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      contact_methods
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      contact_methods/email
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/presence
28m 46s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      organizations
28m 49s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://contacts/
      extensions
28m 49s  343 TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.android.
```

```
        contacts/raw_contacts

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 64

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                                 API signature
   0m  4s    1 <reset>                                                             TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://sms/
   0m  4s    1 <reset>                                                             TId:    1 android.telephony.TelephonyManager: java.lang.String getLinelNumber()
   0m  4s    1 <reset>                                                             TId:    1 android.telephony.TelephonyManager: java.lang.String
         getSimSerialNumber()
   0m  4s    1 <reset>                                                             TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://settings/system
   0m  4s    1 <reset>                                                             TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://call_log/calls
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/history
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://browser/searches
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://browser/bookmarks
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.browser/history
   0m 13s    3 background                                                          TId:  938 android.telephony.TelephonyManager: java.lang.String
         getSimSerialNumber()
   0m 13s    3 background                                                          TId:  938 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   0m 13s    3 background                                                          TId:  941 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://sms/
   0m 14s    3 background                                                          TId:  941 android.telephony.TelephonyManager: java.lang.String getLinelNumber()
   0m 14s    4 background                                                          TId:  941 java.net.URL: java.net.URLConnection openConnection()
   0m 15s    4 background                                                          TId:  941 java.net.Socket: void <init>
   0m 48s    7 <reset>                                                             TId:  938 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/history
   0m 48s    7 <reset>                                                             TId:  938 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 48s    7 <reset>                                                             TId:  938 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://browser/searches
   0m 48s    7 <reset>                                                             TId:  938 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://browser/bookmarks
   0m 48s    7 <reset>                                                             TId:  938 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.browser/history
  10m 17s  120 <reset>                                                             TId:  990 java.net.Socket: void <init>
  11m 28s  137 unlabeled                                                           TId:    1 android.telephony.TelephonyManager: java.lang.String getLinelNumber()
  17m 49s  211 background                                                          TId: 1022 android.content.ContentResolver: android.database.Cursor query(
        android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.
        chrome.browser/bookmarks
  28m 37s  342 background                                                          TId: 1092 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://com.android.browser/bookmarks
  28m 43s  343 <reset>                                                             TId: 1093 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://com.android.browser/history
```

```
28m 43s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://sms
28m 44s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://logs/historys
28m 44s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: android.database.Cursor query(
      android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://logs/
      historys
28m 44s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://sms/inbox
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://sms/sent
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms/inbox
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms-sms
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms/address
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms/part
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms/sent
28m 45s  343 <reset>                                                    TId: 1094 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://mms/outbox
28m 45s  343 <reset>                                                    TId: 1095 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://call_log/calls
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/contacts
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/data
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/raw_contact_entities
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/emails
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/postals
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/groups
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://com.android.contacts/groups_summary
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/people
28m 45s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/phones
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/photos
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/calls
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/settings
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://contacts/deleted_people
28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
```

```
       .String,java.lang.String[]) uri: content://contacts/people/with_email_or_im_filter
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/groups
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/deleted_groups
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/groupmembership
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/groupmembershipraw
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/contact_methods
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/contact_methods/email
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/presence
  28m 46s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/organizations
  28m 49s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://contacts/extensions
  28m 49s  343 <reset>                                                    TId: 1096 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://com.android.contacts/raw_contacts


================================================================
use-case:scanReport:com.cleanmaster.security
================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'Scan' button.
//  3. Wait until the 'Resolve All' button appears and click it.
//  4. Wait until the 'Report' button appears to report 'AntiVirus' app and click it.
//  5. Wait until the 'OK' button appears to confirm 'AntiVirus' report success and click it.
//  6. Click the 'Finish' button.
//  7. Terminate the exploration.

Total run time:         lm lls
Total actions count:    7 (including the final action terminating exploration)

----------------------------------------------------------------
Unique API calls count observed in the run: 9

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  6s    1 |    0m  5s    1 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
         content://ks.cm.antivirus.config.security
   0m  6s    1 |    0m  5s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.launcher2.settings/favorites
   0m  7s    1 |    0m  6s    1 TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.PhoneStateListener,int)
   0m  7s    1 |    0m  6s    1 TId: 2122 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
       ContentObserver) uri: content://browser/bookmarks
   0m  8s    1 |    0m  6s    1 TId: 2123 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
```

```
         ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 11s     1 |     0m 13s     2 TId: 2121 java.net.URL: java.net.URLConnection openConnection()
   0m  9s     1 |     0m 13s     2 TId: 2121 java.net.Socket: void <init>
   0m 52s    18 |     0m 15s     3 TId:    1 android.os.PowerManager$WakeLock: void acquire()
   0m 56s    18 |     0m 19s     3 TId: 2137 android.os.PowerManager$WakeLock: void release(int)

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 14

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate      | Use case       Event                                                                  API signature
   0m  6s     1 |     0m  5s     1 <reset>                                                              TId:    1 android.content.ContentResolver: android.net.Uri
         insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m  6s     1 |     0m  5s     1 <reset>                                                              TId:    1 android.content.ContentResolver: android.database.
         Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
         ://com.android.launcher2.settings/favorites
   0m  7s     1 |     0m  6s     1 <reset>                                                              TId:    1 android.telephony.TelephonyManager: void listen(
         android.telephony.PhoneStateListener,int)
   0m  7s     1 |     0m  6s     1 <reset>                                                              TId: 2122 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://browser/bookmarks
   0m  8s     1 |     0m  6s     1 <reset>                                                              TId: 2123 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 17s     2 |     0m 13s     2 background                                                           TId: 2121 java.net.URL: java.net.URLConnection openConnection
         ()
   0m 17s     2 |     0m 13s     2 background                                                           TId: 2121 java.net.Socket: void <init>
        None! |     0m 14s     2 click:[res:id/layout_main]                                           TId:    1 android.content.ContentResolver: android.net.Uri
               insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m 22s     4 |     0m 14s     2 background                                                           TId: 2133 android.content.ContentResolver: android.net.Uri
         insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
        None! |     0m 15s     3 click:[res:id/oprator_finish]                                        TId:    1 android.os.PowerManager$WakeLock: void acquire()
  56m 17s   773 |     0m 15s     3 click:[res:id/oprator_finish]                                        TId:    1 android.content.ContentResolver: android.net.Uri
        insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m 56s    18 |     0m 19s     3 background                                                           TId: 2137 android.os.PowerManager$WakeLock: void release(int)
        None! |     0m 22s     4 click:[res:id/dialog_btn_report]                                     TId:    1 android.content.ContentResolver: android.net.Uri
               insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   1m 54s    24 |     0m 50s     6 click:[res:id/finish]                                                 TId:    1 android.content.ContentResolver: android.net.Uri
         insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security


============================================================
DroidMate-run:com.cleanmaster.security
============================================================

Total run time:       120m 24s
Total actions count: 1683 (including the final action terminating exploration)
Total resets count:    99 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 13

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>
```

```
 DroidMate     API signature
   0m  6s    1 TId: 1344 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri: content://ks.cm.
         antivirus.config.security
   0m  6s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.launcher2.settings/favorites
   0m  7s    1 TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.PhoneStateListener,int)
   0m  7s    1 TId: 1354 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://browser/bookmarks
   0m  8s    1 TId: 1354 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://com.android.chrome.browser/bookmarks
   0m 17s    2 TId: 1352 java.net.URL: java.net.URLConnection openConnection()
   0m 17s    2 TId: 1352 java.net.Socket: void <init>
   0m 23s    5 TId: 1367 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://ks.cm.antivirus.firewall.security/call_block_logs
   0m 28s    7 TId: 1369 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://ks.cm.antivirus.firewall.security/user_rules
   0m 35s   10 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.contacts/data/phones
   0m 52s   18 TId:    1 android.os.PowerManager$WakeLock: void acquire()
   0m 56s   18 TId: 1384 android.os.PowerManager$WakeLock: void release(int)
  73m 14s 1007 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
        String[],java.lang.String,android.os.CancellationSignal) uri: content://call_log/calls

---------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 54

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
      API call data>

 DroidMate     Event                                                    API signature
   0m  6s    1 <reset>                                                  TId: 1344 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m  6s    1 <reset>                                                  TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android
         .launcher2.settings/favorites
   0m  7s    1 <reset>                                                  TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   0m  7s    1 <reset>                                                  TId: 1354 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://browser/bookmarks
   0m  8s    1 <reset>                                                  TId: 1354 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.android.chrome.browser/bookmarks
   0m 17s    2 background                                               TId: 1352 java.net.URL: java.net.URLConnection openConnection()
   0m 17s    2 background                                               TId: 1352 java.net.Socket: void <init>
   0m 22s    4 background                                               TId: 1356 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m 23s    5 background                                               TId: 1367 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://ks.cm.
         antivirus.firewall.security/call_block_logs
   0m 23s    5 click:[res:id/menu_item_call_block]                      TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
   0m 28s    7 background                                               TId: 1369 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://ks.cm.
         antivirus.firewall.security/user_rules
   0m 35s   10 click:[res:id/edit_import_contact]                       TId:    1 android.content.ContentResolver: android.database.Cursor query(
```

```
      android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android
      .contacts/data/phones
 0m 46s   16 click:[res:id/custom_title_layout_left]                    TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 0m 51s   18 click:[res:id/btn_scan]                                   TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 0m 52s   18 click:[res:id/btn_scan]                                   TId:    1 android.os.PowerManager$WakeLock: void acquire()
 0m 56s   18 background                                                TId: 1384 android.os.PowerManager$WakeLock: void release(int)
 1m  1s   20 click:[res:id/btnRepair]                                  TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 1m 54s   24 click:[res:id/finish]                                     TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 2m 10s   29 click:[res:id/tv_banner]                                  TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 3m 39s   58 click:[res:id/browser_huoyan_report]                      TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 4m 44s   68 click:[res:id/layout_private_bg]                          TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 4m 52s   71 click:[res:id/details_bottom_right]                       TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 5m  1s   75 click:[res:id/details_app_operate]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 7m 44s  115 click:[res:id/menu_item_rateus]                           TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 9m  5s  130 click:[res:id/dialog_btn_continue]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 9m 31s  133 click:[res:id/menu_item_virus_db_update]                  TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
 9m 43s  134 click:[res:id/dialog_btn_ok]                              TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
10m 10s  143 click:[res:id/details_bottom_left]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
11m 20s  155 click:[res:id/sdscan_back]                                TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
11m 20s  155 click:[res:id/sdscan_back]                                TId:    1 android.os.PowerManager$WakeLock: void release(int)
15m  1s  220 click:[res:id/dialog_btn_open]                           TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
15m  3s  221 click:[res:id/setting_protect_intime_btn]                 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
15m  8s  223 unlabeled                                                TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
15m 10s  224 click:[res:id/setting_auto_update_btn]                    TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
15m 12s  225 click:[res:id/setting_open_inspire_layout]                TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
15m 15s  226 click:[res:id/setting_auto_update_layout]                 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
16m 53s  232 click:[res:id/dialog_timing_week]                         TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
17m  4s  237 click:[res:id/setting_protect_intime_layout]              TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
17m  8s  239 click:[res:id/setting_safe_scan_btn]                      TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
17m 11s  240 click:[res:id/setting_safe_scan_layout]                   TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
      Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
26m  2s  361 click:[res:id/dialog_timing_day]                          TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
```

```
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  26m 19s  369 click:[res:id/dialog_timing_off]                          TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  26m 35s  373 click:[res:id/protect_operator_group]                     TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  27m 17s  384 click:[res:id/layout_state]                               TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  49m 56s  684 click:[res:id/intl_about_feedback]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  54m 16s  736 click:[res:id/setting_open_inspire_btn]                   TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  55m 42s  762 click:[res:id/main_title_btn_back]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  56m 17s  773 click:[res:id/oprator_finish]                             TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  62m 17s  870 click:[res:id/dialog_timing_month]                        TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  73m 14s 1007 click:[res:id/edit_import_calllog]                        TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://call_log/
       calls
  78m 11s 1088 click:[res:id/mLanguageList]                             TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://ks.cm.antivirus.config.security
  80m 31s 1126 <reset>                                                  TId: 2622 java.net.Socket: void <init>
  82m 53s 1165 click:[res:id/main_title_btn_back]                        TId:    1 android.os.PowerManager$WakeLock: void release(int)
 111m 42s 1551 background                                               TId: 3034 android.os.PowerManager$WakeLock: void acquire()


============================================================
use-case:findBySearch:com.ebay.mobile
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'Agree' button to agree to terms of use.
//  3. Click the 'Sign in' button.
//  4. Enter 'debugg7@gmail.com' as user name for sign-in.
//  5. Enter 'qwerfdsal' as password for sign-in.
//  6. Click the 'Sign-in' button to finally sign-in.
//  7. Enter 'pillow' in the search field to search for a pillow on eBay.
//  8. Confirm search.
//  9. Click on the first search result to view the item.
// 10. Terminate the exploration.

Total run time:         0m 52s
Total actions count:   10 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 13

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  6s    1 |    0m  8s    1 TId:    1 android.location.LocationManager: void requestLocationUpdates(java.lang.String,long,float,android.location.
```

```
        LocationListener)
   0m  8s     1 |     0m  8s     1 TId: 2315 java.net.Socket: void <init>
   0m  9s     1 |     0m  9s     1 TId: 2326 java.net.URL: java.net.URLConnection openConnection()
   0m 10s     1 |     0m 10s     2 TId:     1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 10s     1 |     0m 10s     2 TId:     1 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://
         com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 10s     1 |     0m 10s     2 TId:     1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
         content://com.ebay.mobile.providers.itemcacheprovider/name_value
         None! |     0m 29s     7 TId: 2350 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.
                lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
         None! |     0m 30s     7 TId:     1 android.os.PowerManager$WakeLock: void acquire()
         None! |     0m 30s     7 TId: 2315 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
                String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.mobile.providers.itemcacheprovider/
                saved_search
         None! |     0m 30s     7 TId: 2351 android.os.PowerManager$WakeLock: void release(int)
         None! |     0m 30s     7 TId: 2348 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
                String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.mobile.providers.itemcacheprovider/
                local_notifications
         None! |     0m 30s     7 TId: 2363 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://
                com.ebay.mobile.providers.itemcacheprovider/local_notifications
         None! |     0m 32s     7 TId: 2350 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://
                com.ebay.mobile.providers.itemcacheprovider/saved_search

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 21

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                    API signature
   0m  6s     1 |     0m  8s     1 <reset>                                                                TId:     1 android.location.LocationManager: void
         requestLocationUpdates(java.lang.String,long,float,android.location.LocationListener)
   0m  8s     1 |     0m  8s     1 <reset>                                                                TId: 2315 java.net.Socket: void <init>
   0m  9s     1 |     0m  9s     1 <reset>                                                                TId: 2326 java.net.URL: java.net.URLConnection openConnection
         ()
         None! |     0m 10s     2 click:[res:id/accept_btn]                                              TId:     1 android.content.ContentResolver: android.database.
                Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
                content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 31s     4 |     0m 10s     2 background                                                             TId: 2326 java.net.URL: java.net.URLConnection openConnection
         ()
         None! |     0m 10s     2 click:[res:id/accept_btn]                                              TId:     1 android.content.ContentResolver: int delete(android.
                net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
         None! |     0m 10s     2 click:[res:id/accept_btn]                                              TId:     1 android.content.ContentResolver: android.net.Uri
                insert(android.net.Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 30s     4 |     0m 12s     3 background                                                             TId: 2333 java.net.Socket: void <init>
         None! |     0m 29s     7 enterText:[res:id/home_search_bar]                                      TId:     1 android.content.ContentResolver: android.database.
                Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
                content://com.ebay.mobile.providers.itemcacheprovider/name_value
         None! |     0m 29s     7 enterText:[res:id/home_search_bar]                                      TId:     1 android.content.ContentResolver: int delete(android.
                net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
         None! |     0m 29s     7 background                                                             TId: 2350 android.content.ContentResolver: int update(android.
                net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/
                name_value
```

```
       None! |    0m 30s    7 enterText:[res:id/home_search_bar]                          TId:    1 android.content.ContentResolver: android.net.Uri
             insert(android.net.Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
       None! |    0m 30s    7 enterText:[res:id/home_search_bar]                          TId:    1 android.os.PowerManager$WakeLock: void acquire()
       None! |    0m 30s    7 background                                                 TId: 2315 android.content.ContentResolver: android.database.
             Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
             content://com.ebay.mobile.providers.itemcacheprovider/saved_search
  4m 15s   75 |    0m 30s    7 background                                                 TId: 2350 android.content.ContentResolver: android.net.Uri
      insert(android.net.Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
       None! |    0m 30s    7 background                                                 TId: 2351 android.os.PowerManager$WakeLock: void release(int)
       None! |    0m 30s    7 background                                                 TId: 2348 android.content.ContentResolver: android.database.
             Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
             content://com.ebay.mobile.providers.itemcacheprovider/local_notifications
       None! |    0m 30s    7 background                                                 TId: 2363 android.content.ContentResolver: int delete(android.
             net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/local_notifications
  4m 15s   75 |    0m 31s    7 background                                                 TId: 2365 android.content.ContentResolver: android.database.
      Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
      ://com.ebay.mobile.providers.itemcacheprovider/name_value
       None! |    0m 32s    7 background                                                 TId: 2350 android.content.ContentResolver: int delete(android.
             net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/saved_search
  4m 15s   75 |    0m 33s    7 background                                                 TId: 2367 android.content.ContentResolver: int delete(android.
      net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value


============================================================
DroidMate-run:com.ebay.mobile
============================================================

Total run time:       120m 22s
Total actions count: 2011 (including the final action terminating exploration)
Total resets count:    95 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 9

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  6s    1 TId:    1 android.location.LocationManager: void requestLocationUpdates(java.lang.String,long,float,android.location.LocationListener)
   0m  8s    1 TId: 3217 java.net.Socket: void <init>
   0m  9s    1 TId: 3228 java.net.URL: java.net.URLConnection openConnection()
   0m 10s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 10s    1 TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.
         providers.itemcacheprovider/name_value
   0m 10s    1 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
   1m 38s   26 TId:    1 android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
   1m 38s   26 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang.String)
  19m 55s  335 TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.ebay.mobile.
        ebaysearch/suggestions

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 27

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
```

```
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
      API call data>

 DroidMate     Event                                                                  API signature
   0m  6s    1 <reset>                                                              TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
         lang.String,long,float,android.location.LocationListener)
   0m  8s    1 <reset>                                                              TId: 3217 java.net.Socket: void <init>
   0m  9s    1 <reset>                                                              TId: 3228 java.net.URL: java.net.URLConnection openConnection()
   0m 10s    1 <reset>                                                              TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
   0m 10s    1 <reset>                                                              TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 10s    1 <reset>                                                              TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   0m 31s    4 background                                                           TId: 3244 java.net.URL: java.net.URLConnection openConnection()
   1m 38s   26 <reset>                                                              TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   1m 38s   26 <reset>                                                              TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   2m 24s   35 background                                                           TId: 3225 java.net.Socket: void <init>
   4m 15s   75 unlabeled                                                            TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
   4m 15s   75 unlabeled                                                            TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   4m 15s   75 unlabeled                                                            TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   5m 50s  109 click:[res:android:id/checkbox]                                     TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
   5m 51s  109 click:[res:android:id/checkbox]                                     TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
   5m 51s  109 click:[res:android:id/checkbox]                                     TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
  17m 17s  290 click:[res:id/list]                                                TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
  17m 17s  290 click:[res:id/list]                                                TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
  17m 17s  290 click:[res:id/list]                                                TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://com.ebay.mobile.providers.itemcacheprovider/name_value
  19m 55s  335 click:[res:android:id/button1]                                     TId:    1 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.ebay.mobile.ebaysearch/suggestions
  61m 34s 1037 click:[res:id/saveSearch]                                          TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.ebay.
         mobile.providers.itemcacheprovider/name_value
  62m  2s 1048 click:[res:id/register_button]                                     TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
  62m  2s 1048 click:[res:id/register_button]                                     TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
         lang.String,long,float,android.location.LocationListener)
  62m  2s 1048 click:[res:id/register_button]                                     TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
  98m 17s 1652 click:[res:id/menu_search]                                         TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
  98m 17s 1652 click:[res:id/menu_search]                                         TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
```

```
        lang.String,long,float,android.location.LocationListener)
  98m 17s 1652 click:[res:id/menu_search]                                TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
        .String)


============================================================
use-case:killTask:com.estrongs.android.taskmanager
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'OK' button to close the 'what's new in current version' pop-up
//  3. Click the tab with 'Kill All' label to access the menu for selecting tasks to kill.
//  4. Click the 'X' circle button at the first task on the list to kill the task.
//  5. Terminate the exploration.

Total run time:         0m 11s
Total actions count:    5 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 2

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
    0m  3s    1 |    0m  3s    2 TId: 2440 java.net.URL: java.net.URLConnection openConnection()
    0m  9s    2 |    0m  3s    2 TId: 2440 java.net.Socket: void <init>

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 2

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                    API signature
    5m 53s   99 |    0m  3s    2 background                                             TId: 2440 java.net.URL: java.net.URLConnection openConnection
          ()
    0m  9s    2 |    0m  3s    2 background                                             TId: 2440 java.net.Socket: void <init>


============================================================
DroidMate-run:com.estrongs.android.taskmanager
============================================================

Total run time:       120m 17s
Total actions count: 2052 (including the final action terminating exploration)
Total resets count:   130 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 5
```

```
Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  3s    1 TId: 3656 java.net.URL: java.net.URLConnection openConnection()
   0m  9s    2 TId: 3656 java.net.Socket: void <init>
   1m 10s   13 TId: 3652 android.net.wifi.WifiManager: boolean setWifiEnabled(boolean)
   3m 42s   54 TId:    1 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   3m 42s   54 TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.PhoneStateListener,int)

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 21

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
      API call data>

 DroidMate     Event                                                    API signature
   0m  3s    1 <reset>                                                  TId: 3656 java.net.URL: java.net.URLConnection openConnection()
   0m  9s    2 background                                               TId: 3656 java.net.Socket: void <init>
   1m 10s   13 background                                               TId: 3652 android.net.wifi.WifiManager: boolean setWifiEnabled(boolean)
   3m 42s   54 click:[res:id/content_power_optim]                       TId:    1 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   3m 42s   54 click:[res:id/content_power_optim]                       TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   4m 24s   71 click:[res:android:id/button1]                           TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   4m 37s   72 <reset>                                                  TId:    1 android.telephony.TelephonyManager: java.lang.String getDeviceId()
   4m 37s   72 <reset>                                                  TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   5m 11s   85 unlabeled                                                TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   5m 53s   99 background                                               TId: 3731 java.net.URL: java.net.URLConnection openConnection()
   7m 46s  134 click:[res:id/btn_startup_optim]                          TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   8m  0s  135 click:[res:id/content_startup]                            TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
   8m 23s  140 click:[res:id/item_button]                                TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
         PhoneStateListener,int)
  21m  2s  357 unlabeled                                                TId:    1 android.telephony.TelephonyManager: java.lang.String getDeviceId()
  68m  3s 1160 <reset>                                                  TId: 4181 android.net.wifi.WifiManager: boolean setWifiEnabled(boolean)
  77m 38s 1313 click:[res:android:id/select_dialog_listview]            TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
        PhoneStateListener,int)
  80m  2s 1354 click:[res:id/menu_list_item]                            TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
        PhoneStateListener,int)
  81m 59s 1385 click:[res:android:id/text1]                             TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
        PhoneStateListener,int)
  84m 21s 1425 click:[res:id/kill]                                      TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
        PhoneStateListener,int)
  94m 40s 1620 click:[res:id/choose_time_view]                          TId:    1 android.telephony.TelephonyManager: java.lang.String getDeviceId()
  94m 40s 1620 click:[res:id/choose_time_view]                          TId:    1 android.telephony.TelephonyManager: void listen(android.telephony.
        PhoneStateListener,int)


============================================================
use-case:convertCurrency:com.frank_weber.forex2
============================================================
```

```
// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'Let's start' button to close the 'welcome' pop-up.
//  3. Click the '1' digit.
//  4. Click the '5' digit.
//  5. Click the '9' digit.
//  6. Click the 'swap currencies' button. The user now reads the converted currency.
//  7. Terminate the exploration.

Total run time:         0m 13s
Total actions count:    7 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 2

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  1s    1 |    0m  1s    1 TId: 2473 java.net.URL: java.net.URLConnection openConnection()
   0m  1s    1 |    0m  1s    1 TId: 2473 java.net.Socket: void <init>

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 2

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m  1s    1 |    0m  1s    1 <reset>                                                                TId: 2473 java.net.URL: java.net.URLConnection openConnection
         ()
   0m  1s    1 |    0m  1s    1 <reset>                                                                TId: 2473 java.net.Socket: void <init>


============================================================
DroidMate-run:com.frank_weber.forex2
============================================================

Total run time:       120m 19s
Total actions count: 2354 (including the final action terminating exploration)
Total resets count:   113 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 2

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  1s    1 TId:  176 java.net.URL: java.net.URLConnection openConnection()
   0m  1s    1 TId:  176 java.net.Socket: void <init>
```

```
------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 4

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                                API signature
   0m  1s    1 <reset>                                                            TId:  176 java.net.URL: java.net.URLConnection openConnection()
   0m  1s    1 <reset>                                                            TId:  176 java.net.Socket: void <init>
  19m 49s  381 background                                                         TId:  260 java.net.Socket: void <init>
 100m 47s 1983 background                                                         TId:  514 java.net.URL: java.net.URLConnection openConnection()


==================================================================
use-case:searchForJob:com.indeed.android.jobsearch
==================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click 'Use without account' to proceed without an account.
//  3. Click the 'what' (job title) search field.
//  4. Click the 's' letter on the on-screen keyboard to make a suggestion pop-up appear with job titles starting with 's'.
//  5. Click the 'sales' job title in the suggestion box.
//  6. Click the 'where' (city) search field.
//  7. Click the 'n' letter on the on-screen keyboard to make a suggestion pop-up appear with cities starting with 'n'.
//  8. Click the 'New York, NY' entry in the suggestion box.
//  9. Click the 'Find Jobs' button.
// 10. Click the first found job on the list.
// 11. Terminate the exploration.

Total run time:          0m 30s
Total actions count:   11 (including the final action terminating exploration)

------------------------------------------------------------------
Unique API calls count observed in the run: 1

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  3s    1 |    0m  2s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook.katana.provider.AttributionIdProvider

------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 1

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                API signature
```

```
   0m  3s    1 |    0m  2s    1 <reset>                                                                    TId:    1 android.content.ContentResolver: android.database.
         Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
         ://com.facebook.katana.provider.AttributionIdProvider


============================================================
DroidMate-run:com.indeed.android.jobsearch
============================================================

Total run time:       120m 20s
Total actions count: 2392 (including the final action terminating exploration)
Total resets count:    80 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 1

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  3s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook.katana.provider.AttributionIdProvider

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 1

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
      API call data>

 DroidMate     Event                                                                 API signature
   0m  3s    1 <reset>                                                             TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.
         facebook.katana.provider.AttributionIdProvider


============================================================
use-case:addEffect:com.picsart.studio
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'Close' button to close the 'Stretch tool' popup.
//  3. Click the 'Effect' button to add an effect to a photo.
//  4. Click the 'Gallery' button to open a gallery with existing photos.
//  5. Click the first photo in the 'Recent' list of photos.
//  6. Click the 'twilight' effect button.
//  7. Click the 'Apply' button to apply the 'twilight' effect.
//  8. Click the 'Save' button to save the modified photo.
//  9. Click the 'Save to SD Card' button.
// 10. Click the 'OK' button to confirm the save of the modified photo to SD card.
// 11. Terminate the exploration.

Total run time:         1m  3s
Total actions count:   11 (including the final action terminating exploration)
```

```
----------------------------------------------------------------
Unique API calls count observed in the run: 10

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  9s    1 |    0m 11s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
         ContentObserver) uri: content://com.picsart.studio.provider/user.update
   0m 10s    1 |    0m 11s    1 TId:    1 android.location.LocationManager: java.lang.String getBestProvider(android.location.Criteria,boolean)
   0m 10s    1 |    0m 11s    1 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang.String)
   0m 10s    1 |    0m 11s    1 TId: 2714 java.net.URL: java.net.URLConnection openConnection()
   0m 10s    1 |    0m 11s    1 TId: 2713 java.net.Socket: void <init>
   0m 10s    1 |    0m 12s    1 TId:    1 android.location.LocationManager: void requestLocationUpdates(java.lang.String,long,float,android.location.
         LocationListener)
   0m 11s    1 |    0m 12s    1 TId: 2714 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.picsart.studio.provider/notifications
   0m 11s    1 |    0m 13s    1 TId: 2713 android.content.ContentResolver: int bulkInsert(android.net.Uri,android.content.ContentValues[]) uri: content://
         com.picsart.studio.provider/notifications
        None! |    0m 29s    6 TId: 2710 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
               String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.providers.media.documents/document/
               image%3A3836
        None! |    0m 29s    6 TId: 2710 android.content.ContentResolver: java.io.InputStream openInputStream(android.net.Uri) uri: content://com.android.
               providers.media.documents/document/image%3A3836

----------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 17

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                    API signature
   0m  9s    1 |    0m 11s    1 <reset>                                                                  TId:    1 android.content.ContentResolver: void
         registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
   0m 10s    1 |    0m 11s    1 <reset>                                                                  TId:    1 android.location.LocationManager: java.lang.String
         getBestProvider(android.location.Criteria,boolean)
   0m 10s    1 |    0m 11s    1 <reset>                                                                  TId:    1 android.location.LocationManager: boolean
         isProviderEnabled(java.lang.String)
   0m 10s    1 |    0m 11s    1 <reset>                                                                  TId: 2714 java.net.URL: java.net.URLConnection openConnection
         ()
   0m 10s    1 |    0m 11s    1 <reset>                                                                  TId: 2713 java.net.Socket: void <init>
   0m 10s    1 |    0m 12s    1 <reset>                                                                  TId:    1 android.location.LocationManager: void
         requestLocationUpdates(java.lang.String,long,float,android.location.LocationListener)
   0m 11s    1 |    0m 12s    1 <reset>                                                                  TId: 2714 android.content.ContentResolver: android.database.
         Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
         ://com.picsart.studio.provider/notifications
   0m 11s    1 |    0m 13s    1 <reset>                                                                  TId: 2713 android.content.ContentResolver: int bulkInsert(
         android.net.Uri,android.content.ContentValues[]) uri: content://com.picsart.studio.provider/notifications
  18m 59s  295 |    0m 16s    3 click:[res:id/start_fx_id]                                               TId:    1 android.location.LocationManager: java.lang.String
        getBestProvider(android.location.Criteria,boolean)
   0m 15s    3 |    0m 16s    3 background                                                               TId: 2693 java.net.URL: java.net.URLConnection openConnection
         ()
```

```
   0m 15s    3 |    0m 16s    3 background                                                    TId: 2693 java.net.Socket: void <init>
   0m 15s    3 |    0m 17s    3 background                                                    TId: 2694 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://com.picsart.studio.provider/notifications
       None! |    0m 23s    4 click:[res:id/galleryButtonId]                                 TId:    1 android.content.ContentResolver: void
              registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
       None! |    0m 23s    4 background                                                    TId: 2694 android.content.ContentResolver: int bulkInsert(
              android.net.Uri,android.content.ContentValues[]) uri: content://com.picsart.studio.provider/notifications
       None! |    0m 29s    6 background                                                    TId: 2710 android.content.ContentResolver: android.database.
              Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
              content://com.android.providers.media.documents/document/image%3A3836
       None! |    0m 29s    6 background                                                    TId: 2710 android.content.ContentResolver: java.io.InputStream
               openInputStream(android.net.Uri) uri: content://com.android.providers.media.documents/document/image%3A3836
   0m 23s    6 |    0m 29s    6 unlabeled                                                     TId:    1 android.location.LocationManager: java.lang.String
       getBestProvider(android.location.Criteria,boolean)


============================================================
DroidMate-run:com.picsart.studio
============================================================

Total run time:       120m 25s
Total actions count: 1910 (including the final action terminating exploration)
Total resets count:   107 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 13

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  9s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
         content://com.picsart.studio.provider/user.update
   0m 10s    1 TId:    1 android.location.LocationManager: java.lang.String getBestProvider(android.location.Criteria,boolean)
   0m 10s    1 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang.String)
   0m 10s    1 TId:  214 java.net.URL: java.net.URLConnection openConnection()
   0m 10s    1 TId:  215 java.net.Socket: void <init>
   0m 10s    1 TId:    1 android.location.LocationManager: void requestLocationUpdates(java.lang.String,long,float,android.location.LocationListener)
   0m 11s    1 TId:  194 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.picsart.studio.provider/notifications
   0m 11s    1 TId:  214 android.content.ContentResolver: int bulkInsert(android.net.Uri,android.content.ContentValues[]) uri: content://com.picsart.studio
         .provider/notifications
   2m 15s   30 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook.katana.provider.AttributionIdProvider
   2m 54s   39 TId:  222 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://com.picsart.studio.provider/notifications
   3m  3s   43 TId:  244 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://com.picsart.studio
         .provider/notifications
   8m 36s  125 TId:    1 android.hardware.Camera: android.hardware.Camera open()
   8m 36s  125 TId:    1 android.hardware.Camera: android.hardware.Camera open(int)

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 69

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
```

```
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
      API call data>

 DroidMate     Event                                                    API signature
   0m  9s    1 <reset>                                                  TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
   0m 10s    1 <reset>                                                  TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   0m 10s    1 <reset>                                                  TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   0m 10s    1 <reset>                                                  TId:  214 java.net.URL: java.net.URLConnection openConnection()
   0m 10s    1 <reset>                                                  TId:  215 java.net.Socket: void <init>
   0m 10s    1 <reset>                                                  TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
         lang.String,long,float,android.location.LocationListener)
   0m 11s    1 <reset>                                                  TId:  194 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.picsart
         .studio.provider/notifications
   0m 11s    1 <reset>                                                  TId:  214 android.content.ContentResolver: int bulkInsert(android.net.Uri,
         android.content.ContentValues[]) uri: content://com.picsart.studio.provider/notifications
   0m 15s    3 click:[res:id/whats_new_closeButton]                     TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   0m 15s    3 background                                               TId:  214 java.net.URL: java.net.URLConnection openConnection()
   0m 15s    3 background                                               TId:  214 java.net.Socket: void <init>
   0m 15s    3 background                                               TId:  194 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.picsart
         .studio.provider/notifications
   0m 46s    8 click:[res:id/start_camera_id]                           TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
   0m 57s   11 click:[res:id/save_photo_btn]                            TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
   1m 51s   19 click:[res:id/start_shop_id]                             TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
   1m 56s   21 click:[res:id/shop_login_button]                         TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   1m 56s   21 click:[res:id/shop_login_button]                         TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   1m 56s   21 click:[res:id/shop_login_button]                         TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
         lang.String,long,float,android.location.LocationListener)
   2m  8s   27 click:[res:id/profile_signup_btn]                        TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   2m  8s   27 click:[res:id/profile_signup_btn]                        TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   2m  8s   27 click:[res:id/profile_signup_btn]                        TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
         lang.String,long,float,android.location.LocationListener)
   2m 15s   30 unlabeled                                                TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.
         facebook.katana.provider.AttributionIdProvider
   2m 25s   33 unlabeled                                                TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   2m 54s   39 background                                               TId:  222 android.content.ContentResolver: int update(android.net.Uri,android.
         content.ContentValues,java.lang.String,java.lang.String[]) uri: content://com.picsart.studio.provider/notifications
   3m  3s   43 background                                               TId:  244 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://com.picsart.studio.provider/notifications
   3m  5s   44 click:[dsc:Notifications, Navigate up]                   TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
         android.location.Criteria,boolean)
   3m  9s   46 click:[res:id/start_edit_id]                             TId:    1 android.content.ContentResolver: void registerContentObserver(android
```

```
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
 3m 14s   48 click:[res:id/picsInGalleryLayoutId]                          TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
 3m 14s   48 click:[res:id/picsInGalleryLayoutId]                          TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
      .String)
 3m 14s   48 click:[res:id/picsInGalleryLayoutId]                          TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
      lang.String,long,float,android.location.LocationListener)
 3m 50s   54 click:[res:id/start_poplar_photo_id]                          TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
 3m 52s   55 click:[dsc:PicsArt, Navigate up]                              TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
 5m  0s   74 l-click:[dsc:More]                                            TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
 7m  6s  106 click:[res:id/shop_item_buy_btn]                              TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
 7m  6s  106 click:[res:id/shop_item_buy_btn]                              TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
      .String)
 7m  6s  106 click:[res:id/shop_item_buy_btn]                              TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
      lang.String,long,float,android.location.LocationListener)
 8m 36s  125 unlabeled                                                     TId:    1 android.hardware.Camera: android.hardware.Camera open()
 8m 36s  125 unlabeled                                                     TId:    1 android.hardware.Camera: android.hardware.Camera open(int)
 8m 48s  130 unlabeled                                                     TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
 9m 21s  136 click:[dsc:Shop, Navigate home]                               TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
11m  0s  166 click:[dsc:Shop, Navigate up]                                TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
12m 10s  184 click:[res:id/custom_effects]                                 TId:    1 android.hardware.Camera: android.hardware.Camera open()
12m 10s  184 click:[res:id/custom_effects]                                 TId:    1 android.hardware.Camera: android.hardware.Camera open(int)
13m 20s  203 click:[res:id/start_collage_id]                               TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
14m  5s  215 click:[res:id/shop_item_buy_install_button]                   TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
14m  5s  215 click:[res:id/shop_item_buy_install_button]                   TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
      .String)
14m  5s  215 click:[res:id/shop_item_buy_install_button]                   TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
      lang.String,long,float,android.location.LocationListener)
14m  8s  216 click:[res:id/si_ui_socialin_sign_fb]                         TId:    1 android.content.ContentResolver: android.database.Cursor query(
      android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook
      .katana.provider.AttributionIdProvider
16m 10s  252 click:[res:id/delete_photo_btn]                               TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
16m 46s  262 click:[dsc:More]                                             TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
16m 49s  263 click:[dsc:Refresh]                                          TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
18m 59s  295 click:[res:id/start_fx_id]                                   TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
23m 41s  367 click:[dsc:Search]                                           TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
23m 53s  370 click:[res:id/picasa_item_image_layout]                      TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
23m 57s  371 click:[res:id/menu_item_close]                               TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
      android.location.Criteria,boolean)
26m 27s  413 click:[res:id/whats_new_videoIcon]                           TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
```

```
  29m 27s  459 click:[dsc:Refresh]                                          TId:    1 android.content.ContentResolver: void registerContentObserver(android
       .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
  31m  7s  489 unlabeled                                                    TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
       .String)
  31m  7s  489 unlabeled                                                    TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
       lang.String,long,float,android.location.LocationListener)
  33m 45s  532 l-click:[dsc:Notifications]                                  TId:    1 android.content.ContentResolver: void registerContentObserver(android
       .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
  46m 40s  747 click:[res:android:id/text1]                                  TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook
       .katana.provider.AttributionIdProvider
  56m  7s  877 click:[res:id/info_dialog_right_button_id]                   TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
       android.location.Criteria,boolean)
  60m 18s  943 click:[dsc:Navigate up]                                      TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
       android.location.Criteria,boolean)
  61m 12s  961 click:[txt:PicsArt]                                          TId:    1 android.location.LocationManager: java.lang.String getBestProvider(
       android.location.Criteria,boolean)
  61m 12s  961 click:[txt:PicsArt]                                          TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
       .String)
  61m 12s  961 click:[txt:PicsArt]                                          TId:    1 android.location.LocationManager: void requestLocationUpdates(java.
       lang.String,long,float,android.location.LocationListener)
  66m 58s 1048 click:[res:id/start_draw_id]                                 TId:    1 android.content.ContentResolver: void registerContentObserver(android
       .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
  67m 37s 1057 click:[res:id/whats_new_closeButton]                         TId:    1 android.content.ContentResolver: void registerContentObserver(android
       .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update
 110m 40s 1757 click:[dsc:Notifications]                                    TId:    1 android.content.ContentResolver: void registerContentObserver(android
       .net.Uri,boolean,android.database.ContentObserver) uri: content://com.picsart.studio.provider/user.update


============================================================
use-case:viewAndCreateDir:com.rhmsoft.fm
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the first directory on the list of dirs to view its contents.
//  3. Click the 'Create' button to setup directory creation action.
//  4. Click the the 'Folder' button to request creation of directory, not file.
//  5. Enter the name of the new dir: 'temp_utc'.
//  6. Click the 'OK' button to confirm the creation of the 'temp_utc' dir.
//  7. Terminate the exploration.

Total run time:         0m 22s
Total actions count:    7 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 0

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature

------------------------------------------------------------
```

```
Unique [API call, event] pairs count observed in the run: 0

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                          API signature


============================================================
DroidMate-run:com.rhmsoft.fm
============================================================

Total run time:       120m 38s
Total actions count: 1876 (including the final action terminating exploration)
Total resets count:   150 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 11

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   4m  0s   28 TId: 2513 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://media/external/images/media
   4m  0s   28 TId: 2513 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://media/external/images/thumbnails
  15m  8s  198 TId: 2597 android.net.wifi.WifiManager$WifiLock: void acquire()
  15m  8s  198 TId: 2597 android.os.PowerManager$WakeLock: void acquire()
  15m  9s  198 TId: 2598 java.net.Socket: void <init>
  15m 14s  200 TId: 2597 android.net.wifi.WifiManager$WifiLock: void release()
  15m 14s  200 TId: 2597 android.os.PowerManager$WakeLock: void release(int)
  21m 31s  313 TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://media/external/
         file
  21m 31s  313 TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://media/external/
         images/media
  21m 31s  313 TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://media/external/
         audio/media
  21m 31s  313 TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://media/external/
         video/media

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 13

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                          API signature
   4m  0s   28 background                                                     TId: 2513 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/
         external/images/media
   4m  0s   28 background                                                     TId: 2513 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/
```

```
        external/images/thumbnails
   4m  3s   29 unlabeled                                                    TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/
         external/images/media
  15m  8s  198 background                                                  TId: 2597 android.net.wifi.WifiManager$WifiLock: void acquire()
  15m  8s  198 background                                                  TId: 2597 android.os.PowerManager$WakeLock: void acquire()
  15m  9s  198 background                                                  TId: 2598 java.net.Socket: void <init>
  15m 14s  200 background                                                  TId: 2597 android.net.wifi.WifiManager$WifiLock: void release()
  15m 14s  200 background                                                  TId: 2597 android.os.PowerManager$WakeLock: void release(int)
  21m 31s  313 background                                                  TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://media/external/file
  21m 31s  313 background                                                  TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://media/external/images/media
  21m 31s  313 background                                                  TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://media/external/audio/media
  21m 31s  313 background                                                  TId: 2600 android.content.ContentResolver: int delete(android.net.Uri,java.lang
        .String,java.lang.String[]) uri: content://media/external/video/media
  31m 30s  467 <reset>                                                     TId: 2649 java.net.Socket: void <init>


============================================================
use-case:edit_friend:com.snapchat.android
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'LOG IN' button on the landing page (first screen).
//  3. Enter the user name.
//  4. Enter the password.
//  5. Click the 'LOG IN' button to actually log into the app.
//  6. Open the 'my friends' list by clicking the 'menu' button in the lower right corner.
//  7. Click the search button in upper right to search for friends.
//  8. Search for 'abc' by entering text into the auto-selected text field that appeared in upper left.
//  9. Click on the displayed 'abc' name.
// 10. Click on the cogwheel next to the friend name to display a menu allowing to add him.
// 11. Click the 'Add friend' button in the settings popup.
// 12. Click on the displayed 'abc' name.
// 13. Click on the cogwheel next to the friend name to display a menu allowing to block him.
// 14. Click the 'Block' button in the settings popup.
// 15. Click on the cogwheel next to the friend name to display a menu allowing to unblock him.
// 16. Click the 'Unblock' button in the settings popup.
// 17. Click on the cogwheel next to the friend name to display a menu allowing to delete him.
// 18. Click the 'Delete' button in the settings popup.
// 19. Terminate the exploration.

Total run time:         1m  8s
Total actions count:   19 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 3

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>
```

```
 DroidMate     | Use case       API signature
   0m  5s    1 |    0m 21s    5 TId: 1176 java.net.Socket: void <init>
   0m 25s    5 |    0m 23s    5 TId: 1179 android.hardware.Camera: android.hardware.Camera open(int)
   0m 27s    5 |    0m 24s    5 TId: 1185 java.net.URL: java.net.URLConnection openConnection()

---------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 3

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m 23s    5 |    0m 21s    5 background                                                           TId: 1176 java.net.Socket: void <init>
   0m 25s    5 |    0m 23s    5 background                                                           TId: 1179 android.hardware.Camera: android.hardware.Camera
         open(int)
   0m 27s    5 |    0m 24s    5 background                                                           TId: 1185 java.net.URL: java.net.URLConnection openConnection
        ()


===============================================================
use-case:find_friends:com.snapchat.android
===============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'LOG IN' button on the landing page (first screen).
//  3. Enter the user name.
//  4. Enter the password.
//  5. Click the 'LOG IN' button to actually log into the app.
//  6. Open the 'my friends' list by clicking the 'menu' button in the lower right corner.
//  7. Click the 'add friend' button in the upper right corner.
//  8. Show the 'contacts list' tab by clicking on the 'contacts notebook' tab icon, in the upper right part of the screen.
//  9. Press the 'back' button.
// 10. Click the 'add friend' button in the upper right corner.
// 11. Allow access to contacts list by clicking the 'Allow Access' button.
// 12. Terminate the exploration.

Total run time:         0m 58s
Total actions count:   12 (including the final action terminating exploration)

---------------------------------------------------------------
Unique API calls count observed in the run: 4

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  5s    1 |    0m 22s    5 TId: 1173 java.net.Socket: void <init>
   0m 25s    5 |    0m 25s    5 TId: 1176 android.hardware.Camera: android.hardware.Camera open(int)
   0m 27s    5 |    0m 27s    5 TId: 1178 java.net.URL: java.net.URLConnection openConnection()
   5m 20s  111 |    0m 56s   11 TId: 1179 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.contacts/data/phones
```

```
---------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 4

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m 23s    5 |    0m 22s    5 background                                                             TId: 1173 java.net.Socket: void <init>
   0m 25s    5 |    0m 25s    5 background                                                             TId: 1176 android.hardware.Camera: android.hardware.Camera
          open(int)
   0m 27s    5 |    0m 27s    5 background                                                             TId: 1178 java.net.URL: java.net.URLConnection openConnection
          ()
   5m 20s  111 |    0m 56s   11 background                                                             TId: 1179 android.content.ContentResolver: android.database.
         Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
         ://com.android.contacts/data/phones


===============================================================
use-case:take_snap:com.snapchat.android
===============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'LOG IN' button on the landing page (first screen).
//  3. Enter the user name.
//  4. Enter the password.
//  5. Click the 'LOG IN' button to actually log into the app.
//  6. Click the 'take snap' button in the lower middle to make a picture with a camera.
//  7. Click the screen (displaying the taken snap) to make the caption edit field appear.
//  8. Add a caption to the taken snap.
//  9. Press the 'back' button.
// 10. Click the stopwatch in the lower left corner to set time.
// 11. Set snap retention time of 4 seconds (instead of default 3) by clicking in lower part of the time spinner menu, being displayed at the bottom.
// 12. Press the 'back' button.
// 13. Click the right arrow in the bottom right to show a list of snap recipients.
// 14. Check-mark myself as the sole snap recipient.
// 15. Send the snap by clicking the right arrow in the lower right of the screen.
// 16. View the snap that just arrived (because I sent it to myself) by clicking on its status in its entry in the snap feed list.
// 17. Terminate the exploration.

Total run time:         1m 23s
Total actions count:   17 (including the final action terminating exploration)

---------------------------------------------------------------
Unique API calls count observed in the run: 4

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  5s    1 |    0m 21s    5 TId: 1169 java.net.Socket: void <init>
```

```
   0m 25s    5 |    0m 24s    5 TId: 1172 android.hardware.Camera: android.hardware.Camera open(int)
   0m 27s    5 |    0m 26s    5 TId: 1171 java.net.URL: java.net.URLConnection openConnection()
   4m 54s  100 |    1m 21s   16 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/external/images/media

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 4

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m 23s    5 |    0m 21s    5 background                                                           TId: 1169 java.net.Socket: void <init>
   0m 25s    5 |    0m 24s    5 background                                                           TId: 1172 android.hardware.Camera: android.hardware.Camera
         open(int)
   0m 27s    5 |    0m 26s    5 background                                                           TId: 1171 java.net.URL: java.net.URLConnection openConnection
         ()
        None! |    1m 21s   16 l-click:[res:id/status]                                              TId:    1 android.content.ContentResolver: android.database.
              Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri:
              content://media/external/images/media


============================================================
use-case:take_video_snap:com.snapchat.android
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'LOG IN' button on the landing page (first screen).
//  3. Enter the user name.
//  4. Enter the password.
//  5. Click the 'LOG IN' button to actually log into the app.
//  6. Click and hold the 'take snap' button in the lower middle to make a video with a camera.
//  7. Click the 'pencil' button in the upper right corner to start drawing on the device.
//  8. Pick a color by clicking in the center of the color palette that just appeared in the upper right corner.
//  9. Draw a line across the screen (displaying the taken video snap).
// 10. Save the video to gallery by clicking the 'save' button in lower left.
// 11. Click the 'add to story' button in lower left.
// 12. Click 'add' in the pop-up box to confirm I want to add the snap to my story.
// 13. Terminate the exploration.

Total run time:         1m  2s
Total actions count:   13 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 5

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  5s    1 |    0m 21s    5 TId: 1181 java.net.Socket: void <init>
```

```
   0m 25s    5 |    0m 23s    5 TId: 1184 android.hardware.Camera: android.hardware.Camera open(int)
   0m 27s    5 |    0m 24s    5 TId: 1190 java.net.URL: java.net.URLConnection openConnection()
   4m 42s   97 |    0m 27s    6 TId:    1 android.media.MediaRecorder: void setAudioSource(int)
   4m 42s   97 |    0m 27s    6 TId:    1 android.media.MediaRecorder: void setVideoSource(int)

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 5

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                API signature
   0m 23s    5 |    0m 21s    5 background                                                         TId: 1181 java.net.Socket: void <init>
   0m 25s    5 |    0m 23s    5 background                                                         TId: 1184 android.hardware.Camera: android.hardware.Camera
         open(int)
   0m 27s    5 |    0m 24s    5 background                                                         TId: 1190 java.net.URL: java.net.URLConnection openConnection
         ()
   4m 42s   97 |    0m 27s    6 l-click:[res:id/camera_take_snap_button]                           TId:    1 android.media.MediaRecorder: void setAudioSource(int
         )
   4m 42s   97 |    0m 27s    6 l-click:[res:id/camera_take_snap_button]                           TId:    1 android.media.MediaRecorder: void setVideoSource(int
         )


============================================================
DroidMate-run:com.snapchat.android
============================================================

Total run time:       210m 22s
Total actions count: 4358 (including the final action terminating exploration)
Total resets count:   149 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 11

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m 23s    5 TId: 4608 java.net.Socket: void <init>
   0m 25s    5 TId: 4611 android.hardware.Camera: android.hardware.Camera open(int)
   0m 25s    5 TId:    1 android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
   0m 27s    5 TId: 4617 java.net.URL: java.net.URLConnection openConnection()
   0m 36s    9 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang.String)
   4m 42s   97 TId:    1 android.media.MediaRecorder: void setAudioSource(int)
   4m 42s   97 TId:    1 android.media.MediaRecorder: void setVideoSource(int)
   4m 54s  100 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://media/external/images/media
   5m 20s  111 TId: 4623 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.contacts/data/phones
   9m 14s  190 TId: 4641 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri: content://media/
         external/images/media
   9m 16s  191 TId: 4641 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://media/external/
         images/media/<number>
```

```
-----------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 33

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                                 API signature
   0m 23s    5 background                                                          TId: 4608 java.net.Socket: void <init>
   0m 25s    5 background                                                          TId: 4611 android.hardware.Camera: android.hardware.Camera open(int)
   0m 25s    5 click:[res:id/login_button]                                         TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   0m 27s    5 background                                                          TId: 4617 java.net.URL: java.net.URLConnection openConnection()
   0m 36s    9 click:[res:id/camera_take_snap_button]                              TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   0m 36s    9 click:[res:id/camera_take_snap_button]                              TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   1m 47s   35 <reset>                                                             TId: 4609 android.hardware.Camera: android.hardware.Camera open(int)
   1m 47s   35 <reset>                                                             TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   1m 47s   35 <reset>                                                             TId: 4610 java.net.Socket: void <init>
   4m 42s   97 l-click:[res:id/camera_take_snap_button]                            TId:    1 android.media.MediaRecorder: void setAudioSource(int)
   4m 42s   97 l-click:[res:id/camera_take_snap_button]                            TId:    1 android.media.MediaRecorder: void setVideoSource(int)
   4m 54s  100 unlabeled                                                           TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/
         external/images/media
   5m 20s  111 background                                                          TId: 4623 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android
         .contacts/data/phones
   6m  7s  125 <reset>                                                             TId: 4621 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android
         .contacts/data/phones
   7m 51s  162 click:[res:id/drawing_btn]                                         TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   8m  4s  168 l-click:[res:id/camera_take_snap_button]                           TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
         .String)
   8m  4s  168 l-click:[res:id/camera_take_snap_button]                           TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   8m  7s  169 click:[res:id/snap_preview_relative_layout]                        TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   8m 12s  171 click:[res:id/unmuted_button]                                      TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   8m 17s  173 click:[res:id/toggle_caption_btn]                                  TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   9m 12s  189 click:[res:android:id/button2]                                     TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   9m 14s  190 background                                                        TId: 4641 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://media/external/images/media
   9m 16s  191 background                                                        TId: 4641 android.content.ContentResolver: int delete(android.net.Uri,java.lang
         .String,java.lang.String[]) uri: content://media/external/images/media/<number>
  13m 33s  281 click:[res:id/picture_send_pic]                                    TId:    1 android.location.LocationManager: android.location.Location
        getLastKnownLocation(java.lang.String)
  15m 30s  326 click:[res:android:id/button1]                                     TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
        .String)
  18m 37s  394 click:[res:id/picture_x]                                          TId:    1 android.location.LocationManager: android.location.Location
        getLastKnownLocation(java.lang.String)
```

```
  23m 27s  488 click:[res:id/picture_save_pic]                                TId:    1 android.location.LocationManager: android.location.Location
        getLastKnownLocation(java.lang.String)
  23m 37s  492 click:[res:id/picture_caption]                                 TId:    1 android.location.LocationManager: android.location.Location
        getLastKnownLocation(java.lang.String)
  27m 44s  586 click:[res:id/settings_smart_filters]                          TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
        .String)
  33m  5s  699 click:[res:id/time_picker_button]                              TId:    1 android.location.LocationManager: android.location.Location
        getLastKnownLocation(java.lang.String)
  62m 33s 1327 click:[res:id/settings_smart_filters_checkbox]                 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.lang
        .String)
  65m 58s 1393 <reset>                                                       TId:    1 android.content.ContentResolver: android.database.Cursor query(
        android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/
        external/images/media
 127m 31s 2667 click:[res:id/story_button]                                   TId:    1 android.location.LocationManager: android.location.Location
       getLastKnownLocation(java.lang.String)


================================================================
use-case:searchForProduct:de.barcoo.android
================================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the 'side menu' button to open it.
//  3. Click the 'suchen' ('search') button in the side menu.
//  4. Enter 'pillow' in the search box.
//  5. Press the 'Done' button on the displayed native keyboard to confirm the search.
//  6. Click the first item in the search result list.
//  7. Terminate the exploration.

Total run time:         0m 26s
Total actions count:    7 (including the final action terminating exploration)

----------------------------------------------------------------
Unique API calls count observed in the run: 10

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  4s    1 |    0m  5s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.provider.Setting/setting
   0m  5s    1 |    0m  5s    1 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
         content://de.barcoo.provider.Setting/setting
   0m  5s    1 |    0m  5s    1 TId: 3014 android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
   0m  4s    1 |    0m  5s    1 TId: 3016 java.net.URL: java.net.URLConnection openConnection()
   0m  5s    1 |    0m  6s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook.katana.provider.AttributionIdProvider
   0m  4s    1 |    0m  6s    1 TId: 3016 java.net.Socket: void <init>
   0m 17s    4 |    0m 19s    6 TId:    1 android.location.LocationManager: void requestSingleUpdate(android.location.Criteria,android.app.PendingIntent)
   0m 48s   14 |    0m 23s    6 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
         String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.provider.RememberedProduct/product
   0m 48s   14 |    0m 23s    6 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
         content://de.barcoo.provider.RememberedProduct/product
```

```
   0m 50s   14 |    0m 24s    6 TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.
          lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 14

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m  4s    1 |    0m  5s    1 <reset>                                                                TId:    1 android.content.ContentResolver: android.database.
          Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
          ://de.barcoo.provider.Setting/setting
   0m  5s    1 |    0m  5s    1 <reset>                                                                TId:    1 android.content.ContentResolver: android.net.Uri
          insert(android.net.Uri,android.content.ContentValues) uri: content://de.barcoo.provider.Setting/setting
   0m  5s    1 |    0m  5s    1 <reset>                                                                TId: 3014 android.location.LocationManager: android.location.
          Location getLastKnownLocation(java.lang.String)
   0m  4s    1 |    0m  5s    1 <reset>                                                                TId: 3016 java.net.URL: java.net.URLConnection openConnection
          ()
   0m  5s    1 |    0m  6s    1 <reset>                                                                TId:    1 android.content.ContentResolver: android.database.
          Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
          ://com.facebook.katana.provider.AttributionIdProvider
   0m  4s    1 |    0m  6s    1 <reset>                                                                TId: 3016 java.net.Socket: void <init>
   0m 12s    2 |    0m  8s    2 background                                                             TId: 3014 android.location.LocationManager: android.location.
          Location getLastKnownLocation(java.lang.String)
   0m 14s    3 |    0m  8s    2 background                                                             TId: 3023 java.net.URL: java.net.URLConnection openConnection
          ()
   0m 16s    4 |    0m  9s    2 background                                                             TId: 3016 java.net.Socket: void <init>
   0m 37s   11 |    0m 19s    6 unlabeled                                                              TId:    1 android.location.LocationManager: android.location.
          Location getLastKnownLocation(java.lang.String)
         None! |    0m 19s    6 unlabeled                                                              TId:    1 android.location.LocationManager: void
                  requestSingleUpdate(android.location.Criteria,android.app.PendingIntent)
   0m 48s   14 |    0m 23s    6 unlabeled                                                              TId:    1 android.content.ContentResolver: android.database.
          Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
          ://de.barcoo.provider.RememberedProduct/product
   0m 48s   14 |    0m 23s    6 unlabeled                                                              TId:    1 android.content.ContentResolver: android.net.Uri
          insert(android.net.Uri,android.content.ContentValues) uri: content://de.barcoo.provider.RememberedProduct/product
   0m 50s   14 |    0m 24s    6 unlabeled                                                              TId:    1 android.content.ContentResolver: int update(android.
          net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product


============================================================
DroidMate-run:de.barcoo.android
============================================================

Total run time:       120m 31s
Total actions count: 1605 (including the final action terminating exploration)
Total resets count:   138 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 10

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>
```

```
 DroidMate     API signature
   0m  4s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.provider.Setting/setting
   0m  5s    1 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri: content://de.barcoo.
         provider.Setting/setting
   0m  5s    1 TId: 2992 android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
   0m  5s    1 TId: 2994 java.net.URL: java.net.URLConnection openConnection()
   0m  5s    1 TId: 2994 java.net.Socket: void <init>
   0m  5s    1 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://com.facebook.katana.provider.AttributionIdProvider
   0m 17s    4 TId:    1 android.location.LocationManager: void requestSingleUpdate(android.location.Criteria,android.app.PendingIntent)
   0m 48s   14 TId:    1 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
         String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.provider.RememberedProduct/product
   0m 48s   14 TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri: content://de.barcoo.
         provider.RememberedProduct/product
   0m 50s   14 TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://de.barcoo.provider.RememberedProduct/product

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 38

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                    API signature
   0m  4s    1 <reset>                                                  TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
         provider.Setting/setting
   0m  5s    1 <reset>                                                  TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://de.barcoo.provider.Setting/setting
   0m  5s    1 <reset>                                                  TId: 2992 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   0m  5s    1 <reset>                                                  TId: 2994 java.net.URL: java.net.URLConnection openConnection()
   0m  5s    1 <reset>                                                  TId: 2994 java.net.Socket: void <init>
   0m  5s    1 <reset>                                                  TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.
         facebook.katana.provider.AttributionIdProvider
   0m 12s    2 background                                               TId: 2992 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   0m 14s    3 background                                               TId: 3002 java.net.URL: java.net.URLConnection openConnection()
   0m 16s    4 background                                               TId: 2994 java.net.Socket: void <init>
   0m 17s    4 click:[res:id/smallImageItemTemplate]                    TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   0m 17s    4 click:[res:id/smallImageItemTemplate]                    TId:    1 android.location.LocationManager: void requestSingleUpdate(android.
         location.Criteria,android.app.PendingIntent)
   0m 26s    7 click:[res:id/menuItemTemplate]                          TId:    1 android.location.LocationManager: android.location.Location
         getLastKnownLocation(java.lang.String)
   0m 48s   14 unlabeled                                                TId:    1 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
         provider.RememberedProduct/product
   0m 48s   14 unlabeled                                                TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
         Uri,android.content.ContentValues) uri: content://de.barcoo.provider.RememberedProduct/product
   0m 50s   14 unlabeled                                                TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
         content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
```

```
 1m 21s   17 click:[res:id/fullWidthImageItemTemplate]                    TId:    1 android.location.LocationManager: android.location.Location
       getLastKnownLocation(java.lang.String)
 1m 58s   24 click:[res:id/contentPlaceholder]                            TId:    1 android.location.LocationManager: android.location.Location
       getLastKnownLocation(java.lang.String)
 6m  3s   82 click:[res:id/menuItemTemplate]                              TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
21m 42s  303 click:[res:id/lin_widget44]                                  TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
21m 42s  303 click:[res:id/lin_widget44]                                  TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
21m 42s  303 click:[res:id/lin_widget44]                                  TId:    1 android.location.LocationManager: android.location.Location
       getLastKnownLocation(java.lang.String)
27m 26s  370 click:[dsc:close drawer]                                     TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
27m 26s  370 click:[dsc:close drawer]                                     TId:    1 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://de.barcoo.provider.RememberedProduct/product
27m 29s  371 click:[res:id/menuItemTemplate]                              TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
39m 58s  536 click:[dsc:close drawer]                                     TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
43m 20s  578 click:[res:android:id/default_activity_button]               TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
43m 20s  578 click:[res:android:id/default_activity_button]               TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
46m 30s  611 click:[dsc:Navigate up]                                      TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
46m 30s  611 click:[dsc:Navigate up]                                      TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
60m 20s  799 click:[res:android:id/expand_activities_button]              TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
60m 20s  799 click:[res:android:id/expand_activities_button]              TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
64m 39s  854 click:[res:id/title]                                         TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
64m 39s  854 click:[res:id/title]                                         TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
77m 13s 1024 l-click:[res:android:id/default_activity_button]             TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
77m 13s 1024 l-click:[res:android:id/default_activity_button]             TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
111m 49s 1500 click:[res:id/history_gridview]                             TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://de.barcoo.
       provider.RememberedProduct/product
111m 49s 1500 click:[res:id/history_gridview]                             TId:    1 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://de.barcoo.provider.RememberedProduct/product
111m 49s 1500 click:[res:id/history_gridview]                             TId:    1 android.location.LocationManager: android.location.Location
       getLastKnownLocation(java.lang.String)
```

```
============================================================
use-case:openUrl:org.mozilla.firefox
============================================================

// Manually-written description of the actions of the use case:
//
//  1. Reset the app by calling Package Manager through adb (Android Debug Bridge).
//  2. Click the address bar.
//  3. Enter 'www.google.com' in the address bar.
//  4. Click the right arrow ('Go') button to go to google.com.
//  5. Terminate the exploration.

Total run time:         0m 25s
Total actions count:    5 (including the final action terminating exploration)

------------------------------------------------------------
Unique API calls count observed in the run: 10

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API in DroidMate run for the first time, if any> <index of action that triggered the call, if any> | <time of logging the
       unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     | Use case       API signature
   0m  3s    1 |    0m  3s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.
          ContentObserver) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m  3s    1 |    0m  4s    1 TId: 3076 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
          String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m  4s    1 |    0m  5s    2 TId: 3076 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
          String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/combined
   0m  4s    1 |    0m  5s    2 TId: 3079 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
          String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/thumbnails
   0m  4s    1 |    0m  6s    2 TId: 3065 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
          String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/history
   0m  4s    1 |    0m  6s    2 TId: 3065 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
          String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/favicons
   0m  6s    1 |    0m 11s    3 TId: 3068 java.net.Socket: void <init>
   4m 11s   78 |    0m 18s    4 TId: 3065 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.
          lang.String[]) uri: content://org.mozilla.firefox.db.browser/history
   4m 15s   79 |    0m 21s    4 TId: 3065 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.
          lang.String[]) uri: content://org.mozilla.firefox.db.browser/favicons
   4m 16s   79 |    0m 21s    4 TId: 3065 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.
          lang.String[]) uri: content://org.mozilla.firefox.db.browser/thumbnails

------------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 11

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] in DroidMate run for the first time, if any> <index of action that triggered the
       call, if any> | <time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call>
        <the event data> <the API call data>

 DroidMate     | Use case       Event                                                                  API signature
   0m  3s    1 |    0m  3s    1 <reset>                                                               TId:    1 android.content.ContentResolver: void
          registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m  3s    1 |    0m  4s    1 <reset>                                                               TId: 3076 android.content.ContentResolver: android.database.
```

```
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/bookmarks
  0m 16s    4 |    0m  5s    2 background                                                    TId: 3076 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/combined
  0m 16s    4 |    0m  5s    2 background                                                    TId: 3079 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/thumbnails
  2m  3s   40 |    0m  6s    2 background                                                    TId: 3065 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/history
  2m  3s   40 |    0m  6s    2 background                                                    TId: 3065 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/favicons
  0m 16s    4 |    0m  7s    3 background                                                    TId: 3065 android.content.ContentResolver: android.database.
       Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content
       ://org.mozilla.firefox.db.browser/bookmarks
  0m 13s    3 |    0m 11s    3 background                                                    TId: 3068 java.net.Socket: void <init>
  4m 11s   78 |    0m 18s    4 background                                                    TId: 3065 android.content.ContentResolver: int update(android.
       net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/history
  4m 15s   79 |    0m 21s    4 background                                                    TId: 3065 android.content.ContentResolver: int update(android.
       net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/favicons
  4m 16s   79 |    0m 21s    4 background                                                    TId: 3065 android.content.ContentResolver: int update(android.
       net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/thumbnails


============================================================
DroidMate-run:org.mozilla.firefox
============================================================

Total run time:       120m 21s
Total actions count: 2126 (including the final action terminating exploration)
Total resets count:    74 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 17

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call data>

 DroidMate     API signature
   0m  3s    1 TId:    1 android.content.ContentResolver: void registerContentObserver(android.net.Uri,boolean,android.database.ContentObserver) uri:
       content://org.mozilla.firefox.db.browser/bookmarks
   0m  3s    1 TId:  973 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
       String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m  4s    1 TId:  973 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
       String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/combined
   0m  4s    1 TId:  976 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
       String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/thumbnails
   0m  4s    1 TId:  962 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
       String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/history
   0m  4s    1 TId:  962 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
       String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/favicons
   0m  6s    1 TId:  978 java.net.Socket: void <init>
   0m  9s    2 TId:  982 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String[],java.lang.
       String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.browser/combined
```

```
   0m 16s    4 TId:  962 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://org.mozilla.firefox.db.browser/bookmarks
   2m 57s   60 TId:  971 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://org.mozilla.
        firefox.db.browser/history/old
   3m 45s   74 TId:  999 java.net.URL: java.net.URLConnection openConnection()
   4m 11s   78 TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://org.mozilla.firefox.db.browser/history
   4m 15s   79 TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://org.mozilla.firefox.db.browser/favicons
   4m 16s   79 TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
         : content://org.mozilla.firefox.db.browser/thumbnails
  17m  2s  313 TId: 1067 android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.
        String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.tabs/tabs
  22m 16s  404 TId: 1091 android.content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri: content://org.mozilla.
        firefox.db.browser/bookmarks
  69m 35s 1242 TId: 1271 android.content.ContentResolver: int update(android.net.Uri,android.content.ContentValues,java.lang.String,java.lang.String[]) uri
        : content://org.mozilla.firefox.db.browser/bookmarks/parents

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 26

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action that triggered the call> <the event data> <the
       API call data>

 DroidMate     Event                                                    API signature
   0m  3s    1 <reset>                                                  TId:    1 android.content.ContentResolver: void registerContentObserver(android
         .net.Uri,boolean,android.database.ContentObserver) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m  3s    1 <reset>                                                  TId:  973 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/bookmarks
   0m  4s    1 <reset>                                                  TId:  973 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/combined
   0m  4s    1 <reset>                                                  TId:  976 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/thumbnails
   0m  4s    1 <reset>                                                  TId:  962 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/history
   0m  4s    1 <reset>                                                  TId:  962 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/favicons
   0m  6s    1 <reset>                                                  TId:  978 java.net.Socket: void <init>
   0m  9s    2 background                                               TId:  982 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.firefox.db.
         browser/combined
   0m 13s    3 background                                               TId:  965 java.net.Socket: void <init>
   0m 16s    4 background                                               TId:  962 android.content.ContentResolver: int update(android.net.Uri,android.
         content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/bookmarks
   0m 16s    4 background                                               TId:  962 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/bookmarks
   0m 16s    4 background                                               TId:  983 android.content.ContentResolver: android.database.Cursor query(
         android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
         .firefox.db.browser/combined
```

```
 0m 16s    4 background                                                    TId:  984 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
       .firefox.db.browser/thumbnails
 2m  3s   40 background                                                    TId:  971 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
       .firefox.db.browser/history
 2m  3s   40 background                                                    TId:  971 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla
       .firefox.db.browser/favicons
 2m 57s   60 <reset>                                                      TId:  971 android.content.ContentResolver: int delete(android.net.Uri,java.lang
       .String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/history/old
 3m 45s   74 background                                                   TId:  999 java.net.URL: java.net.URLConnection openConnection()
 4m 11s   78 background                                                   TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/history
 4m 15s   79 background                                                   TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/favicons
 4m 16s   79 background                                                   TId:  998 android.content.ContentResolver: int update(android.net.Uri,android.
       content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/thumbnails
17m  2s  313 background                                                   TId: 1067 android.content.ContentResolver: android.database.Cursor query(
      android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://org.mozilla.
      firefox.db.tabs/tabs
17m 59s  330 background                                                   TId: 1073 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/history/old
22m 16s  404 background                                                   TId: 1091 android.content.ContentResolver: int delete(android.net.Uri,java.lang
      .String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/bookmarks
23m  9s  421 <reset>                                                      TId: 1091 android.content.ContentResolver: int update(android.net.Uri,android.
      content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/thumbnails
69m 11s 1234 click:[res:id/restart]                                       TId:    1 android.content.ContentResolver: void registerContentObserver(android
      .net.Uri,boolean,android.database.ContentObserver) uri: content://org.mozilla.firefox.db.browser/bookmarks
69m 35s 1242 background                                                   TId: 1271 android.content.ContentResolver: int update(android.net.Uri,android.
      content.ContentValues,java.lang.String,java.lang.String[]) uri: content://org.mozilla.firefox.db.browser/bookmarks/parents
```

Appendix D

Snapchat comparison summaries

This appendix contains the DROIDMATE exploration summaries and comparison of the two SNAPCHAT versions: 4.1.07 and 5.0.34.6.

- First comes version SNAPCHAT 5.0.34.6, followed by SNAPCHAT 4.1.07.
- The summary format is analogous to the exploration summaries listed in Appendix C, but with following changes:
  - There are no use cases, only automated DROIDMATE explorations.
  - Instead of "Use case" and "DroidMate" columns, we now have "Other" and "This" columns. "Other" denotes the time the API call or *(event, API call)* pair was observed in the other version of SNAPCHAT (or "None" if it wasn't observed at all).

```
============================================================
DroidMate-run:com.snapchat.android-5.0.34.6
============================================================

Total run time:       210m 22s
Total actions count: 4214 (including the final action terminating exploration)
Total resets count:   147 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 13

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call
     data>

 Other          | This           API signature
   0m  5s    1 |    0m  7s    1 TId:  326 java.net.Socket: void <init>
   0m 25s    5 |    0m  7s    1 TId:  328 android.hardware.Camera: android.hardware.Camera open(int)
   0m 27s    5 |    0m 26s    5 TId:  340 java.net.URL: java.net.URLConnection openConnection()
         None! |    0m 37s    7 TId:  374 android.media.AudioRecord: void <init>
   9m 14s  190 |    1m  1s   17 TId:  360 android.content.ContentResolver: android.net.Uri insert(android.net.
        Uri,android.content.ContentValues) uri: content://media/external/images/media
         None! |    1m  2s   17 TId:  360 android.content.ContentResolver: android.database.Cursor query(
              android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.
              os.CancellationSignal) uri: content://media/external/images/thumbnails
         None! |    1m  2s   17 TId:  360 android.content.ContentResolver: android.os.ParcelFileDescriptor
              openFileDescriptor(android.net.Uri,java.lang.String,android.os.CancellationSignal) uri: content
              ://media/external/images/thumbnails/<number>
         None! |    1m  2s   17 TId:  360 android.content.ContentResolver: android.net.Uri insert(android.net.
              Uri,android.content.ContentValues) uri: content://media/external/images/thumbnails
   0m 36s    9 |    2m 44s   51 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.
        lang.String)
         None! |    3m  7s   61 TId:    1 android.telephony.TelephonyManager: java.lang.String getLine1Number
              ()
   5m 20s  111 |    6m  9s  124 TId:  384 android.content.ContentResolver: android.database.Cursor query(
        android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.
        CancellationSignal) uri: content://com.android.contacts/data/phones
         None! |   53m  7s 1094 TId:    1 android.os.PowerManager$WakeLock: void acquire(long)
         None! |   53m  7s 1094 TId:  515 android.os.PowerManager$WakeLock: void release(int)

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 43
```

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action
that triggered the call> <the event data> <the API call data>

```
 Other          | This           Event                                                                  API signature
   0m  5s    1 |    0m  7s    1 <reset>                                                                TId:  326 java.
       net.Socket: void <init>
   1m 47s   35 |    0m  7s    1 <reset>                                                                TId:  328 android
       .hardware.Camera: android.hardware.Camera open(int)
   0m 23s    5 |    0m 25s    5 background                                                             TId:  351 java.
       net.Socket: void <init>
   0m 27s    5 |    0m 26s    5 background                                                             TId:  340 java.
       net.URL: java.net.URLConnection openConnection()
        None! |    0m 37s    7 background                                                             TId:  374 android
             .media.AudioRecord: void <init>
   0m 25s    5 |    0m 39s    8 background                                                             TId:  328 android
       .hardware.Camera: android.hardware.Camera open(int)
   9m 14s  190 |    1m  1s   17 background                                                             TId:  360 android
       .content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
       content://media/external/images/media
        None! |    1m  2s   17 background                                                             TId:  360 android
             .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.
             lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://
             media/external/images/thumbnails
        None! |    1m  2s   17 background                                                             TId:  360 android
             .content.ContentResolver: android.os.ParcelFileDescriptor openFileDescriptor(android.net.Uri,java
             .lang.String,android.os.CancellationSignal) uri: content://media/external/images/thumbnails/<
             number>
        None! |    1m  2s   17 background                                                             TId:  360 android
             .content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues)
             uri: content://media/external/images/thumbnails
        None! |    2m 44s   51 click:[res:id/settings_manage_additional_services]                     TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |    3m  7s   61 click:[res:id/mobile_number]                                           TId:    1 android
             .telephony.TelephonyManager: java.lang.String getLine1Number()
   5m 20s  111 |    6m  9s  124 background                                                             TId:  384 android
       .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
       String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.
       contacts/data/phones
  15m 30s  326 |   26m 31s  549 click:[res:android:id/button1]                                         TId:    1 android
      .location.LocationManager: boolean isProviderEnabled(java.lang.String)
```

```
0m 36s    9 |   26m 52s  551 click:[res:id/camera_take_snap_button]                    TId:    1 android
     .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   27m  8s  558 click:[res:id/camera_my_friends_button]                  TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   28m 34s  587 click:[res:id/checkbox]                                  TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   32m 11s  663 click:[res:id/add_friends_back_button_area]             TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   32m 58s  678 click:[res:id/video_settings_button]                    TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   38m 37s  792 click:[res:id/camera_activity_layout]                   TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   39m  6s  804 unlabeled                                               TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   40m  7s  823 click:[res:id/my_friends_list]                          TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   45m 51s  941 click:[res:id/snap_preview_relative_layout]             TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   48m 51s 1005 click:[res:id/settings_filters_checkbox]                TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   49m 30s 1014 click:[res:id/my_friends_list_item]                     TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   53m  7s 1094 click:[res:id/send_to_bottom_panel_send_button]         TId:    1 android
           .os.PowerManager$WakeLock: void acquire(long)
      None! |   53m  7s 1094 background                                              TId:  515 android
           .os.PowerManager$WakeLock: void release(int)
      None! |   59m 36s 1225 click:[res:id/picture_x]                                TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   67m 21s 1379 click:[res:id/camera_feed_button]                       TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   68m 42s 1398 click:[res:id/story_button]                             TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   80m  6s 1624 click:[res:id/camera_switch_camera]                     TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   91m  1s 1852 click:[res:id/settings_filters]                         TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |  118m  9s 2403 click:[res:id/tabsLayout]                               TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |  125m 52s 2557 click:[res:id/myfriends_action_bar_search_button]       TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |  131m 44s 2672 click:[res:id/my_friends_back_button_area]              TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
```

```
        None! |  133m 13s 2705 click:[res:id/picture_save_pic]                              TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  139m 44s 2831 click:[res:android:id/text1]                                 TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  141m  6s 2859 click:[res:id/myfriends_action_bar_friend_button]           TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  144m 13s 2921 click:[res:id/drawing_btn]                                   TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  171m 59s 3464 click:[res:id/send_to_list]                                  TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  187m 10s 3755 click:[res:id/feed_back_button_area]                         TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  190m  1s 3815 click:[res:id/feed_logo]                                     TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)
        None! |  194m  8s 3897 <reset>                                                      TId:    1 android
             .location.LocationManager: boolean isProviderEnabled(java.lang.String)


============================================================
DroidMate-run:com.snapchat.android-4.1.07
============================================================

Total run time:       210m 22s
Total actions count: 4358 (including the final action terminating exploration)
Total resets count:   149 (including the initial action)

------------------------------------------------------------
Unique API calls count observed in the run: 11

Below follows a list of first calls to unique APIs. It is to be read as follows:
<time of logging the unique API for the first time> <index of action that triggered the call> <the API call
     data>

 Other         | This           API signature
   0m  7s    1 |    0m 23s    5 TId: 4608 java.net.Socket: void <init>
   0m  7s    1 |    0m 25s    5 TId: 4611 android.hardware.Camera: android.hardware.Camera open(int)
         None! |    0m 25s    5 TId:    1 android.location.LocationManager: android.location.Location
              getLastKnownLocation(java.lang.String)
   0m 26s    5 |    0m 27s    5 TId: 4617 java.net.URL: java.net.URLConnection openConnection()
   2m 44s   51 |    0m 36s    9 TId:    1 android.location.LocationManager: boolean isProviderEnabled(java.
       lang.String)
         None! |    4m 42s   97 TId:    1 android.media.MediaRecorder: void setAudioSource(int)
```

```
       None! |    4m 42s   97 TId:    1 android.media.MediaRecorder: void setVideoSource(int)
  0m  7s    1 |    4m 54s  100 TId:    1 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.
       CancellationSignal) uri: content://media/external/images/media
  6m  9s  124 |    5m 20s  111 TId: 4623 android.content.ContentResolver: android.database.Cursor query(
       android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String,android.os.
       CancellationSignal) uri: content://com.android.contacts/data/phones
  1m  1s   17 |    9m 14s  190 TId: 4641 android.content.ContentResolver: android.net.Uri insert(android.net.
       Uri,android.content.ContentValues) uri: content://media/external/images/media
       None! |    9m 16s  191 TId: 4641 android.content.ContentResolver: int delete(android.net.Uri,java.
            lang.String,java.lang.String[]) uri: content://media/external/images/media/<number>

------------------------------------------------------------
Unique [API call, event] pairs count observed in the run: 33

Below follows a list of first calls to unique [API call, event] pairs. It is to be read as follows:
<time of logging the unique API call from the unique [API call, event] for the first time> <index of action
     that triggered the call> <the event data> <the API call data>

 Other         | This          Event                                                                  API signature
  0m 25s    5 |    0m 23s    5 background                                                           TId: 4608 java.
       net.Socket: void <init>
  0m 39s    8 |    0m 25s    5 background                                                           TId: 4611 android
       .hardware.Camera: android.hardware.Camera open(int)
       None! |    0m 25s    5 click:[res:id/login_button]                                          TId:    1 android
            .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
  0m 26s    5 |    0m 27s    5 background                                                           TId: 4617 java.
       net.URL: java.net.URLConnection openConnection()
 26m 52s  551 |    0m 36s    9 click:[res:id/camera_take_snap_button]                                TId:    1 android
      .location.LocationManager: boolean isProviderEnabled(java.lang.String)
       None! |    0m 36s    9 click:[res:id/camera_take_snap_button]                                TId:    1 android
            .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
  0m  7s    1 |    1m 47s   35 <reset>                                                              TId: 4609 android
       .hardware.Camera: android.hardware.Camera open(int)
       None! |    1m 47s   35 <reset>                                                              TId:    1 android
            .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
  0m  7s    1 |    1m 47s   35 <reset>                                                              TId: 4610 java.
       net.Socket: void <init>
       None! |    4m 42s   97 l-click:[res:id/camera_take_snap_button]                             TId:    1 android
            .media.MediaRecorder: void setAudioSource(int)
       None! |    4m 42s   97 l-click:[res:id/camera_take_snap_button]                             TId:    1 android
            .media.MediaRecorder: void setVideoSource(int)
```

```
      None! |    4m 54s  100 unlabeled                                                    TId:    1 android
           .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.
           lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://
           media/external/images/media
 6m  9s  124 |    5m 20s  111 background                                                   TId: 4623 android
     .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
     String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com.android.
     contacts/data/phones
      None! |    6m  7s  125 <reset>                                                      TId: 4621 android
           .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.
           lang.String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://com
           .android.contacts/data/phones
      None! |    7m 51s  162 click:[res:id/drawing_btn]                                   TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |    8m  4s  168 l-click:[res:id/camera_take_snap_button]                     TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |    8m  4s  168 l-click:[res:id/camera_take_snap_button]                     TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |    8m  7s  169 click:[res:id/snap_preview_relative_layout]                  TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |    8m 12s  171 click:[res:id/unmuted_button]                                TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |    8m 17s  173 click:[res:id/toggle_caption_btn]                            TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |    9m 12s  189 click:[res:android:id/button2]                               TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
 1m  1s   17 |    9m 14s  190 background                                                   TId: 4641 android
     .content.ContentResolver: android.net.Uri insert(android.net.Uri,android.content.ContentValues) uri:
     content://media/external/images/media
      None! |    9m 16s  191 background                                                   TId: 4641 android
           .content.ContentResolver: int delete(android.net.Uri,java.lang.String,java.lang.String[]) uri:
           content://media/external/images/media/<number>
      None! |   13m 33s  281 click:[res:id/picture_send_pic]                              TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
26m 31s  549 |   15m 30s  326 click:[res:android:id/button1]                              TId:    1 android
     .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   18m 37s  394 click:[res:id/picture_x]                                     TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |   23m 27s  488 click:[res:id/picture_save_pic]                              TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |   23m 37s  492 click:[res:id/picture_caption]                               TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
```

```
      None! |   27m 44s  586 click:[res:id/settings_smart_filters]                        TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
      None! |   33m  5s  699 click:[res:id/time_picker_button]                            TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
      None! |   62m 33s 1327 click:[res:id/settings_smart_filters_checkbox]               TId:    1 android
           .location.LocationManager: boolean isProviderEnabled(java.lang.String)
0m  7s    1 |   65m 58s 1393 <reset>                                                     TId:    1 android
     .content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.
     String,java.lang.String[],java.lang.String,android.os.CancellationSignal) uri: content://media/external
     /images/media
      None! |  127m 31s 2667 click:[res:id/story_button]                                 TId:    1 android
           .location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)
```