SPECK: From Google Textual Guidelines to Automatic Detection of Android Apps Vulnerabilities

Mauro Conti, *Senior Member, IEEE,* Eleonora Losiouk, and Roberto Rossini

Abstract—The success of the Android OS is due to its open source nature and its support towards developers to publish their own apps. While being an open platform is a benefit, it also requires a reliable security model to protect mobile users from attacks. To address this problem, Google published a set of textual guidelines describing how to prevent security and privacy issues.
In this paper, we "translate" the Google guidelines into 32 rules and propose SPECK: a rule-based static taint analysis system able to analyze both the source code of an app (to support its developers) and the APK file of an app (to provide a security report to its users). We manually verified the statistical precision of our tool in checking the rules against the apps (23 rules have a precision greater than 80%) and used them to analyze the top 100 popular apps on the Play Store. We found that each app has at least one violation, while more than the 50% of the apps violates at least 17 rules. Few rules are violated by almost all the apps, while the 90.13% of violations is located in external libraries. The developers more prone to errors are X-Flow (developer of Happy Color™), Voodoo (developer of Fire Balls 3D, Ball Mayhem!, Hole.io, Paper.io 2, Helix Jump) and Full Fat (developer of Grass cut).

Index Terms—Mobile Applications; Mobile code security; Security and Privacy Protection

1 Introduction

Mobile Operating System (OS) are complex platforms, that involve different actors, from mobile OS vendors to mobile device manufacturers. Having a market share equal to 72,18% [1], Android is the leading mobile OS worldwide and it has been designed as an open platform, encouraging developers to publish their own apps on the Google Play Store by paying a small subscription. At the same time, mobile users rely on the Google Play Store as a market place to choose the apps to be installed on their mobile devices. Managing an open platform means also having a robust security model, which provides mobile users a secure ecosystem of apps and devices. Among several ones, Google relies on two approaches to protect its platform: (i) the Android Security Program; (ii) the cloud-based services for monitoring apps. The Android Security Program aims at identifying the vulnerabilities of the Android platform in order to release the appropriate security patches. On the other side, the cloud-based services aim at monitoring apps published on the Google Play Store or installed on mobile users devices. The Android Security Program involves four phases: (i) vulnerability reporting; (ii) security patch development; (iii) vulnerability and patch notification; (iv) patch deployment. Anyone (e.g., an academic researcher, a member of the Android security team, a device manufacturer) can identify a security issue and responsibly disclose it to Google. The Android security team, then, takes care of the security patch development or of the integration of any proposed solution with the Android Open Source Project (AOSP). Once ready, the Android device manufacturers are notified, so that they can integrate the patch into their custom Android OS. Concerning the cloud-based services, Google provides the following ones: *Google Play*, as an app market that also provides community review, app license verification and app security scanning; *Android updates to mobile devices*; *App services*, to allow apps to use cloud capabilities for saving data; *Verify Apps*, to warn about the installation of harmful apps and to scan the ones installed on devices; *SafetyNet*, an intrusion detection system to mitigate security threats; *SafetyNet Attestation*.

The above-mentioned set of tools focuses on malicious Android apps, while those that are not malicious might anyway contain vulnerabilities due to an improper usage of the Android Application Programming Interface (API) or an inadequate knowledge of the Android security issues [2]. To address this problem, Google proposed three solutions: (i) a set of textual guidelines [3], including tips and suggestions for developers to prevent them from introducing security and privacy issues in their own apps; (ii) a code scanning tool called Lint [4], that searches for issues related to correctness, security, performance, usability, accessibility, and internationalization; (iii) a course on the Google Play Academy platform[1], to help developers with implementing the guidelines and adopting a "security by design" approach in their apps. Despite the above-mentioned proposals, we believe there is an urgent need in the Android community of a solution that comprehensively analyzes both an Android

- *M. Conti and E.Losiouk are with the Department of Mathematics, University of Padova, Italy.*
  E-mail: conti@math.unipd.it, eleonora.losiouk@unipd.it
- *R.Rossini is with the Department of Information Engineering, University of Padova, Italy.*
  E-mail: robertorossini96@gmail.com

Manuscript received October 15, 2021.

1. https://playacademy.exceedlms.com/student/path/63550-security-by-design

app source code, thus helping the developers to fix its issues, and Android app compiled code, to provide mobile users a security evaluation of the app they are going to install on their devices.

In this paper, we first analyze the Google security and privacy guidelines, which refer to several vulnerabilities and provide suggestions for developers (in a textual format) to prevent their introduction in the app source code. We considered each vulnerability described by Google and "translated" the suggestion into a rule, aimed at detecting the vulnerability. We ended up with the formalization of 32 rules. For each vulnerability described in the Google guidelines, we also identified the associated attacks that a malicious app can launch against the vulnerable one. We, then, propose SPECK (Security and Privacy chECK of Android apps vulnerabilities), a rule-based static taint analysis system that automatically finds the violations to our rules. In particular, for each violated rule, SPECK shows the developer the specific line of code where the vulnerability has been detected, thus prompting him to fix the issue. SPECK is designed for both developers and users. Developers launch SPECK against their app source code and receive a report on the identified vulnerabilities; mobile users install the SPECK app, through which they request a remote server to analyze a specific app. We manually validated the statistical precision of our tool in checking the rules against the apps (23 rules out of 32 have a precision greater than 80%) and analyzed the Android ecosystem by launching SPECK against 100 popular apps, to find an answer to the following research questions: RQ1 - What is the occurrence of vulnerabilities in Android apps? RQ2 - How long does it take for SPECK to generate an app vulnerability report? RQ3 - What is the origin of the vulnerabilities in Android apps? RQ4 - Which developers are more prone to introduce vulnerabilities in Android apps?. We found that each app has at least one violation to our rules, while more than the 50% of them violates at least 17 rules. Few rules are violated by almost all the apps (some of them even multiple times by the same app). The majority of violations (90.13%) are located in external libraries. The developers more prone to errors are X-Flow (developer of Happy Color™), Voodoo (developer of Fire Balls 3D, Ball Mayhem!, Hole.io, Paper.io 2, Helix Jump) and Full Fat (developer of Grass cut).

Contributions. The contributions of this paper are as follows:

- *Formalization of the Google security and privacy guidelines*: we formalized 32 rules and we manually verified their statistical precision.
- *Vulnerabilities exploitation*: for each vulnerability targeted by a rule, we identified the possible attacks a malicious app can launch against the vulnerable one.
- *SPECK system*: we designed and developed SPECK, a rule-based static taint analysis system that finds violations of our rules in Android apps (the code[2] and demo videos of user mode[3] and developer mode[4] are available online).
- *Analysis of the Android ecosystem*: we used SPECK to analyze the 100 top popular Android apps on the Google Play Store, finding that each one has at least one violation to our rules, while more than the 50% of them violates at least 17 rules.

2. https://github.com/SPRITZ-Research-Group/SPECK
3. https://github.com/SPRITZ-Research-Group/SPECK/blob/main/demo/usermode.gif
4. https://github.com/SPRITZ-Research-Group/SPECK/blob/main/demo/developermode.gif

2 Background

Android apps are written in Java, while native code and shared libraries are developed in C/C++. The bottom layer of the Android architecture is a Linux kernel, specifically customized for embedded environments with limited resources. On the top of the Linux kernel, the native libraries developed in C/C++ support high performance third-party reusable, shared libraries. The Android framework provides the set of Java libraries for app developers.

The Android Application Package (APK) file is a zip archive consisting of several files and folders, where the app is packaged. In particular, the `AndroidManifest.xml` stores the meta-data such as package name, permissions, definitions of one or more components like `Activities`, `Services`, `Broadcast Receivers` or `Content Providers`, minimum and maximum version support, libraries to be linked etc. The executable file `classes.dex` stores the Dalvik bytecode to be executed in the Dalvik Virtual Machine (DVM). As a matter of fact, Android apps are written in Java code, which is then compiled into .class files, an intermediate Java-bytecode. Then, all .class files are merged into a single Dalvik Executable (.dex) file, which is run in the DVM.

The main Android app components are the following ones:

- `Activity`: this is the user interface component of an app, which has to be declared in the `AndroidManifest.xml` file. Apart from some predefined task, an `Activity` can also return the result to its caller. `Activities` are launched using `Intents`.
- `Service`: this component performs background tasks without any UI (e.g., playing an audio or downloading data from the network). `Services` are launched using `Intents`.
- `Broadcast Receiver`: this component listens to the Android system generated events (e.g., SMS_RECEIVED) and to the application-defined events broadcasted by other apps.
- `Content Provider`: this component works as a data-store, that provides an interface for data access for both the app declaring the component and an external app.

Android app has multiple entry-points, according to the number of declared components, which can be invoked or executed independently, since the communication with them is asynchronous. App components are accessible by other apps only if they are explicitly exported.

Android Kernel implements the Linux Discretionary Access Control (DAC). Each app process is assigned a Unique ID (UID) and runs within an isolated sandbox. The sandboxing restrains apps or their system services from interfering among each other. To restrict an app from accessing sensitive resources (e.g., telephony, GPS, network, power-management), Android provides a

permission-based security model in the application framework. Developers must declare the permissions required in `AndroidManifest.xml`. At the install time, if an app has the permissions for accessing a protected resource (e.g., Bluetooth), the app process is assigned to the corresponding Group ID (GID). Thus, apart from UID, each app process may be assigned one or more GID. Android permissions are divided into the following four protection-levels: (i) *normal*, if permissions have a minimal risk on the user, system app or device. Normal permissions are granted by default at the install time; (ii) *dangerous*, if permissions fall within the high risk group due to their capability of accessing the private data and important sensors of the device. A user can grant dangerous permissions at the install time or when the app accesses the protected resources at run-time; (iii) *signature*, only if the app requesting a permission is signed with the same developer certificate of the app that declared that permission; (iv) *SignatureOrSystem*, if the requesting app asking for permissions is signed with the same certificate as the Android system image or with an app that declares these permissions.

3 Related Work

Over the years, researchers provided several static analysis tools aimed at detecting vulnerabilities in Android apps by relying on different approaches: taint analysis, reachability analysis, symbolic execution, APK rewriting, inter-component flow graph, inter-component flow analysis and rule-based analysis. Due to the high amount of sensitive information stored in a mobile device, information leakage is the most addressed concern [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. Other significant issues somehow related to sensitive information leakage are: over-permissioning [9], [11], [12], [18], [25], [26], [27], intent spoofing [10], [28], [29], [30] and unauthorized intent receipt [10], [28], [30]. Finally, other solutions have been proposed to address cryptography misuse [31], [32] and anti-plagiarism detection [33], [34], [35]. In addition to research papers, the Android community has provided several static analysis tools, among which: Lint [4], the official Android code scanning tool, Amandroid [36], Androwarn [37], ApkAnalyser [38], APKInspector [39], APKLeaks [40], apkx [41], BlueSeal [9], [42], CFGScanDroid [43], ClassyShark [44], ConDroid [45], DidFail [10], DroidLegacy [46], DroidRA [47], DroidSafe [6], [48], JAADAS [49], Madrolyzer [50], Quark-Engine [51], RiskInDroid [52], [53], Smali-CFGs [54], SmaliSCA [55], SPARTA [56], StaCoAn [57], SUPER [58]. Table 1 provides an overview of the above-mentioned tools and research papers, specifying the criteria used for finally selecting the ones to be used in the experimental evaluation of SPECK. For each tool, we analyzed the logic to find possible mappings with the SPECK rules, we searched for its public repository and we finally tried to install and run the tool. At the end of this analysis, we identified seven tools that could be compared with SPECK: Argus-Amandroid, Androwarn, Lint, Quark-Engine, RiskInDroid, SUPER and CERT TAPIOCA. For more details about the comparison analysis, please, see Section 6.7.

4 Rules

In this section, we illustrate our formalization of 32 rules, designed through the analysis of the Google security and privacy guidelines [3]. Such guidelines describe several Android vulnerabilities and illustrate how developers should write their apps source code to prevent it from being vulnerable. For each vulnerability described in the guidelines, we formalized a rule aimed at detecting it and we identified the attacks that a malicious app can launch against the vulnerable one. In Table 2, we provide all the 32 rules, which have been classified according to the following Open Web Application Security Project (OWASP) top 10 mobile risks [69]: improper platform usage, insecure data storage, insecure communication, insecure authentication, insufficient cryptography, insecure authorization, client code quality, code tampering, reverse engineering, extraneous functionality.

Out of the 32 designed rules, here we illustrate the five most interesting and violated ones (i.e., Rule 1, Rule 5, Rule 6, Rule 11, and Rule 29). For each rule, we provide first the Google guideline, then the pseudo-code of the rule we formalized and, finally, the set of attacks a malicious app can perform against a vulnerable one. The remaining 27 rules are described in Appendix A. The source code of all the rules is available online[5].

4.1 Rule 1 - Show an app chooser

Google Guideline. *If an implicit intent can launch at least two possible apps on a user's device, explicitly show an app chooser. This interaction strategy allows users to transfer sensitive information to an app that they trust.*

```
Intent intent = new Intent(Intent.ACTION_SEND);
List<ResolveInfo> possibleActivitiesList =
   queryIntentActivities(intent, PackageManager.
   MATCH_ALL);

// Verify that an activity in at least two apps on
// the user's device can handle the intent.
// Otherwise, start the intent only if an app on
// the user's device can handle the intent.
if (possibleActivitiesList.size() > 1) {

  // Create intent to show chooser.
  // Title is something similar to "Share this
  // photo with".

  String title = getResources().getString(R.string.
   chooser_title);
  Intent chooser = Intent.createChooser(intent,
   title);
  startActivity(chooser);
} else if (intent.resolveActivity(
  getPackageManager()) != null) {
  startActivity(intent);
}
```

Listing 1. Show an app chooser

Pseudo-code. The Rule 1 pseudo-code is shown in Algorithm 1.

Attack. The attack aims at intercepting an implicit `Intent`, that is originally sent to a legitimate app, but that is intercepted by a malicious one without any user notification. To complete the attack, a malicious app exploits the implicit

5. https://github.com/SPRITZ-Research-Group/SPECK/tree/main/server/codeAnalysis

TABLE 1
Overview of the state-of-art static analysis tools.

Paper/Tool	Addressed Vulnerabilities	SPECK Rules Overlap	Comment
S. Arzt et al. [5]	-	-	SPECK embeds Flowdroid
M. C. Grace et al. [7]	Risks of in-app advertisement libraries	-	No overlap with SPECK
L. Lu et al. [8]	Component hijacking	-	No public repository
Blueseal [9], [42]	Malware	-	Deprecated
W. Klieber et al. [10]	Inter/intra-component data flows	-	Deprecated
L. Wu et al. [12]	Issues in customized Android images	-	No overlap with SPECK
K. Lu et al. [13]	Privacy leaks	R8, R9, R10, R11, R12	No public repository
X. Xiao et al. [14]	Private data usage and leakage	R8, R9, R10, R11, R12	No public repository
K. O. Elish et al. [15]	Violations to legitimate dataflow patterns	-	No overlap with SPECK
X. Chen et al. [16]	Privacy leaks	R8, R9, R10, R11, R12	No public repository
J. Kim et al. [17]	Privacy leaks	R8, R9, R10, R11, R12	No public repository
X. Cui et al. [18]	Detect privilege escalation vulnerabilities	-	No public repository
J. P. Achara et al. [20]	Privacy implications of the ACCESS_WIFI_STATE permission	-	No public repository
Y. Feng et al. [21]	Malware	-	No overlap with SPECK
M. A. El-Zawawy et al. [22]	Next-intent vulnerabilities	-	No overlap with SPECK
M. A. El-Zawawy et al. [23]	Webview vulnerabilities	-	No overlap with SPECK
A. P. Felt et al. [25]	Overprivileges	-	No public repository
K. W. Y. Au et al. [26]	-	-	No overlap with SPECK
A. Bartel et al. [27]	Detecting permission gaps	R3	No public repository
D. Octeau et al. [28]	Inter-component communication	R1	No public repository
E. Chin et al. [30]	Application communication vulnerabilities	R1	No public repository
S. Fahl t al. [31]	MITM attacks	R5	
M. Egele et al. [59]	Cryptographic misuses	R22, R29, R30	No public repository
J. Chen et al. [33]	-	-	No overlap with SPECK
M. Sun et al. [34]	-	-	No overlap with SPECK
F. Zhang et al. [35]	-	-	No overlap with SPECK
J. Tang et al. [60]	SSL security	R5	No public repository
S. Salva et al. [61]	Intent-based vulnerabilities	R1, R18	Missing files
P. Gadient et al. [62]	Traces for prospect vulnerabilities	R1, R3, R5, R6, R7, R8, R12, R13, R17, R18, R1, R20, R22, R24, R26, R29, R30	It does not analyze apk files
J. Gajrani et al. [63]	Several vulnerabilities	R1, R3, R4, R5, R6, R8, R12, R17, R18, R22, R23, R25, R26, R29	No public repository
H. Shahriar et al. [64]	Content provider leakage vulnerability	R2	No public repository
D. Bassole et al. [65]	Vulnerabilities in the permissions system	R3, R4, R15	No public repository
B. F. Demissie et al. [66]	Permission re-delegation vulnerabilities	R3, R4	No public repository
D. Wu et al. [67]	File:// vulnerabilities	R9	No public repository
Amandroid [36]	Security vetting of Android apps	R5, R22, R29	Comparable with SPECK
Androwarn [37]	Apps malicious behaviours	R21	Comparable with SPECK
ApkAnalyser [38]	-	-	Deprecated
APKInspector [39]	-	-	Deprecated
APKLeaks [40]	URIs, endpoints and secrets	-	No overlap with SPECK
apkx [41]	-	-	No overlap with SPECK
CERT TAPIOCA [68]	MITM	R5	Comparable with SPECK
CFGScanDroid [43]	-	-	No overlap with SPECK
ClassyShark [44]	-	-	No overlap with SPECK
ConDroid [45]	-	-	Deprecated
DidFail [10]	-	-	No overlap with SPECK
DroidRA [47]	-	-	No overlap with SPECK
DroidSafe [6], [48]	Malicious code detection	-	Installation issues
JAADAS [49]	Several vulnerabilities	-	Installation issues
Madrolyzer [50]	Malware detection	-	No overlap with SPECK
Quark-Engine [51]	Several vulnerabilities	R24	Comparable with SPECK
RiskInDroid [52], [53]	Overpermissioning	R3	Comparable with SPECK
Smali-CFGs [54]	-	-	No overlap with SPECK
SmaliSCA [55]	-	-	No overlap with SPECK
SPARTA [56]	Type-checking based malware detection	-	No overlap with SPECK
StaCoAn [57]	Several vulnerabilities	-	Installation issues
SUPER [58]	Several vulnerabilities	R7, R8, R14, R21, R29	Comparable with SPECK

TABLE 2
Rules formalized from the Google security and privacy guidelines.

Rule N	Rule Name	OWASP Mobile Risks
Rule 1	Show an app chooser	Improper Platform Usage
Rule 2	Content provider access control	Improper Platform Usage, Insecure Data Storage
Rule 3	Provide the right permissions	Improper Platform Usage
Rule 4	Use intents to defer permissions	Improper Platform Usage
Rule 5	Use SSL traffic	Insecure Communication
Rule 6	Use HTML message channels	Client Code Quality, Code Tampering
Rule 7	Use WebView objects carefully	Code Tampering
Rule 8	Store private data within internal storage	Insecure Data Storage
Rule 9	Share data securely across apps	Improper Platform Usage, Insecure Communication
Rule 10	Use scoped directory access	Insecure Data Storage
Rule 11	Store only non-sensitive data in cache files	Insecure Data Storage
Rule 12	Use SharedPreferences in private mode	Improper Platform Usage
Rule 13	Keep services and dependencies up-to-date	Insecure Data Storage
Rule 14	Check validity of data	Insecure Data Storage
Rule 15	Create permissions	Improper Platform Usage
Rule 16	Erase data in WebView cache	Insecure Data Storage, Client Code Quality
Rule 17	Avoid SQL injections	Insecure Data Storage, Code Tampering
Rule 18	Prefer explicit intents	Improper Platform Usage
Rule 19	Use IP networking	Insecure Communication, Extraneous Functionality
Rule 20	Use services	Insecure Authentication, Improper Platform Usage
Rule 21	Use telephony networking	Improper Platform Usage, Insecure Communication
Rule 22	Use cryptography	Insufficient Cryptography
Rule 23	Use broadcast receivers	Improper Platform Usage, Insecure Authentication
Rule 24	Dynamically load code	Code Tampering
Rule 25	Common problems with hostname verification	Insecure Communication
Rule 26	Warnings about using SSLSocket directly	Insecure Communication
Rule 27	Configure CAs for debugging	Extraneous Functionality
Rule 28	Opt out of cleartext traffic	Insecure Communication, Code Tampering
Rule 29	Choose a recommended algorithm	Insufficient Cryptography
Rule 30	Deprecated cryptographic functionality	Insufficient Cryptography
Rule 31	Migrate existing data	Insecure Data Storage
Rule 32	Access device encrypted storage	Insecure Data Storage

Rule 1: Show an app chooser

```
begin
    implicitIntents ← getAppImplicitIntents()
    chooserIntents ← getAppChooserIntents()
    foreach cIntent in chooserIntents do
        respected ← False
        foreach intent in implicitIntents do
            if cIntent = intent then
                respected ← True
                break
            end
        end
        if not respected then
            Rule 1 is not respected.
        end
    end
end
```

`Intent` forwarding system of the Android OS and the absence of an app chooser. Thus, by declaring the `Intent Filter` associated to the target implicit `Intent` with the highest priority, the malicious app becomes the recipient of the implicit `Intent`, which will be successfully delivered to the malicious app since no app chooser will be shown.

4.2 Rule 5 - Use SSL traffic

Google Guideline. *If your app communicates with a web server that has a certificate issued by a well-known, trusted CA, the HTTPS request is very simple:*

```
URL url = new URL("https://www.google.com");
HttpsURLConnection urlConnection = (
    HttpsURLConnection) url.openConnection();
urlConnection.connect();
InputStream in = urlConnection.getInputStream();
```

Listing 2. Use SSL traffic

Pseudo-code. The Rule 5 pseudo-code is shown in Algorithm 5.

Attack. The `SSLSocketFactory` can be used to validate the identity of an HTTPS server against a list of trusted certificates and to authenticate to the HTTPS server using a private key.

If HTTPS is not used, or it is used without a validation of the HTTPS server through the `SSLSocketFactory`, a *Man-in-the-Middle* attack can be performed, i.e., an attacker can secretly relay and alter the communication between two parties.

4.3 Rule 6 - Use HTML message channels

Google Guideline. *Because `WebView` consumes web content that can include HTML and JavaScript, improper use can introduce common web security issues such as cross-site-scripting (JavaScript injection). Android includes a number of mechanisms to reduce the scope of these potential issues by limiting the capability of `WebView` to the minimum functionality required by your application.*
If your application doesn't directly use JavaScript within a `WebView`, do not call `setJavaScriptEnabled()`. Some

Rule 5: Use SSL traffic

```
begin
    openConns ← getOpenConnVars()
    httpsOpenConns ← getHttpsOpenConnVars()
    httpsConnSSLs ← getConnSSLSockFactVars()
    foreach openConn in openConns do
        respected ← False
        foreach httpsConn in httpsOpenConns do
            if openConn = httpsConn then
                respected ← True
                break
            end
            if not respected then
                Rule 5 is not respected.
            end
        end
    end
    foreach sslConn in httpsConnSSLs do
        respected ← False
        foreach httpsConn in httpsOpenConns do
            if sslConn = httpsConn then
                respected ← True
                break
            end
            if not respected then
                if not catchesException(sslConn)
                 then
                    Rule 5 is not respected.
                end
            end
        end
    end
end
```

sample code uses this method, which you might repurpose in production application, so remove that method call if it's not required. By default, `WebView` does not execute JavaScript, so cross-site-scripting is not possible.
Use `addJavaScriptInterface()` with particular care because it allows JavaScript to invoke operations that are normally reserved for Android applications. If you use it, expose `addJavaScriptInterface()` only to web pages from which all input is trustworthy. If untrusted input is allowed, untrusted JavaScript may be able to invoke Android methods within your app. In general, we recommend exposing `addJavaScriptInterface()` only to JavaScript that is contained within your application APK.
If your app must use JavaScript interface support on devices running Android 6.0 (API level 23) and higher, use HTML message channels instead of communicate between a website and your app, as shown in the following code snippet:

```
WebView myWebView = (WebView)  findViewById(R.id.
    webview);

// messagePorts[0] and messagePorts[1] represent
// the two ports. They are already tangled to each
// other and have been started.
WebMessagePort[] channel = myWebView.
    createWebMessageChannel();

// Create handler for channel[0] to receive
// messages.
channel[0].setWebMessageCallback(new WebMessagePort
    .WebMessageCallback() {
   @Override
   public void onMessage(WebMessagePort port,
    WebMessage message) {
     Log.d(TAG, "On port " + port + ", received this
     message: " + message);
   }
});

// Send a message from channel[1] to channel[0].
channel[1].postMessage(new WebMessage("My secure
    message"));
```

Listing 3. Use HTML message channels

Pseudo-code. The Rule 6 pseudo-code is shown in Algorithm 6.

Rule 6: Use HTML message channels

```
begin
    s1 ← "setJavaScriptEnabled"
    s2 ← "true"
    arr ← ["evaluateJavascript", "addJavascriptInt-
    erface"]
    methods ← getAllCalledMethods()
    foreach method in methods do
        if method in arr then
            Rule 6 is not respected.
        end
        if method = s1 then
            if getSecondArg(method) = s2 then
                Rule 6 is not respected.
            end
        end
    end
end
```

Attack. An insecure handling of JavaScript code can lead to Cross-Site Scripting (XSS) attacks.

4.4 Rule 11 - Store only non-sensitive data in cache files

Google Guideline. *To provide quicker access to non-sensitive app data, store it in the device's cache. For caches larger than 1 MB in size, use `getExternalCacheDir()`; otherwise, use `getCacheDir()`. Each method provides you with the `File` object that contains your app's cached data.*
The following code snippet shows how to cache a file that your app recently downloaded:

```
File cacheDir = getCacheDir();
File fileToCache = new File(myDownloadedFileUri);
String fileToCacheName = fileToCache.getName();
File cacheFile = new File(cacheDir.getPath(),
    fileToCacheName);
```

Listing 4. Store only non-sensitive data in cache files

***Note**: if you use `getExternalCacheDir()` to place your app's cache within shared storage, the user might eject the media containing this storage while your app is running. You should include logic to gracefully handle the cache miss that this user behavior causes.*
***Caution**: there is no security enforced on these files. Therefore, any app that has the `WRITE_EXTERNAL_STORAGE` permission can access the contents of this cache.*

Pseudo-code. The Rule 11 pseudo-code is shown in Algorithm 11.

Rule 11: Store only non-sensitive data in cache files

```
begin
    arr ← ["getCacheDir", "getExternalCacheDir"]
    methods ← getAllCalledMethods()
    foreach method in methods do
        if method in arr then
            Rule 11 is not respected.
        end
    end
end
```

Attack. A malicious app can access any data saved in the device cache, even the sensitive ones. Moreover, if the legitimate app accesses the cache through the `getExternalCacheDir()` API, it is using an external storage directory accessible by any other app on the same device.

4.5 Rule 29 - Choose a recommended algorithm

Google Guideline. *When you have the freedom to choose which algorithm to use (such as when you do not require compatibility with a third-party system), we recommend using the following algorithms:*

- `Cipher` *class: AES in either CBC or GCM mode with 256-bit keys (such as* `AES/GCM/NoPadding`*)*
- `MessageDigest` *class: SHA-2 family (e.g.,* `SHA-256`*)*
- `Mac` *class: SHA-2 family HMAC (e.g.,* `HMACSHA256`*)*
- `Signature` *class: SHA-2 family with ECDSA (e.g.,* `SHA256withECDSA`*)*

Pseudo-code. The Rule 29 pseudo-code is shown in Algorithm 29.

Rule 29: Choose a recommended algorithm

```
begin
    cryptoMethods ← getAllCryptoMethods()
    foreach method in cryptoMethods do
        if not usesRecommendedClassArgs(method)
         then
            Rule 29 is not respected.
        end
    end
end
```

Attack. If an app does not properly use cryptographic algorithms or it uses insecure ones, a malicious app can break and access to any data or communication, which should have been protected by cryptography.

5 SPECK System

In this section, we describe the design and implementation of SPECK (the code is available online[6]), our proposal that relies on the rules described in Section 4 to automatically detect the vulnerabilities of an Android app. SPECK is a rule-based, static taint analysis system, that encompasses the following three components:

- The *SPECK App* - it lists all the apps installed on a mobile device and allows a user to choose which one will be analyzed by the *SPECK Static Analyzer* on the *SPECK Server*. Once the app has been processed, the *SPECK App* shows the user the final vulnerability report. We developed the *SPECK App* as an Android app.
- The *SPECK Static Analyzer* - it analyzes an Android app by executing the rules we designed from the Google security and privacy guidelines. We developed the *SPECK Static Analyzer* by using the Python programming language. Since some rules (i.e., Rule1, Rule7, Rule9, Rule16, Rule18, Rule22, Rule25, Rule26, Rule30) require a static taint analysis approach to follow the flow of specific data (e.g., `Intent` objects), they rely on the FlowDroid tool [5], [70].
- The *SPECK Server* - it is responsible for fetching an app APK from the *APKPure store*, once the name of the target app is known, or directly from the user device. It extracts the app source code though the *JADX Decompiler* and it, finally, launches the *SPECK Static Analyzer* against the app source code to generate the vulnerability report.

SPECK supports the *User Mode* (the demo video is available online[7]) and the *Developer Mode* (the demo video is available online[8]) shown in Fig. 1 and Fig. 2, respectively. In the first scenario, the user sends through the *SPECK App* an app name to the *SPECK Server* (i.e., Step 1). Then, the *SPECK Server* first checks whether the vulnerability report for that app already exists. If this is the case, the *SPECK Server* immediately returns the existing report. Otherwise, it downloads the apk from the *APKPure store* (i.e., Step 2) and extracts the source code through the *JADX Decompiler* (i.e., Step 3). The *SPECK Server*, then, launches the *SPECK Static Analyzer* (i.e., Step 4) and returns the *Vulnerability Report* to the mobile user (i.e., Step 5 and Step 6). The request sent by the *SPECK App* towards the *SPECK Server* is completely non-blocking and asynchronous, and the user receives a background notification when the *Vulnerability Report* has been downloaded. In the *Developer Mode*, an Android developer launches the *SPECK Static Analyzer* against the source code of his app (i.e., Step 1 and Step 2) and inspects the identified violations specified in the *Vulnerability Report* to fix them afterwards (i.e., Step 3 and Step 4).

6 Results and Evaluation

Here, we illustrate our evaluation of the 32 rules and of the SPECK system. We first describe our experimental setup in Section 6.1 and then the results obtained to answer to our four research questions in Section 6.2, Section 6.3, Section 6.4 and Section 6.5, respectively. In Section 6.6, we provide the precision of the 32 rules, after performing a manual validation of the output returned by SPECK launched against ten

6. https://github.com/SPRITZ-Research-Group/SPECK

7. https://github.com/SPRITZ-Research-Group/SPECK/blob/main/demo/usermode.gif

8. https://github.com/SPRITZ-Research-Group/SPECK/blob/main/demo/developermode.gif

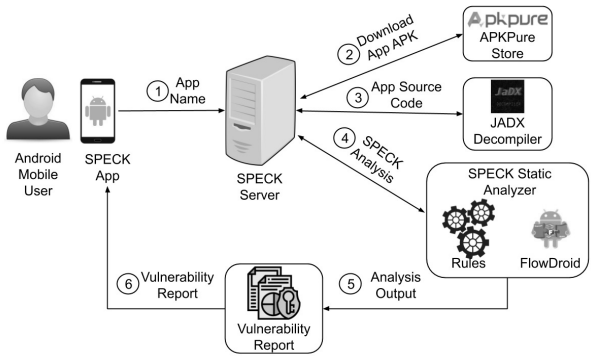

Fig. 1. The SPECK system in *User Mode*.

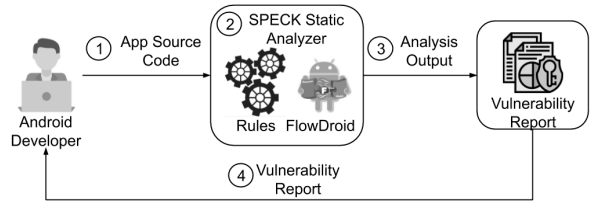

Fig. 2. The SPECK system in *Developer Mode*.

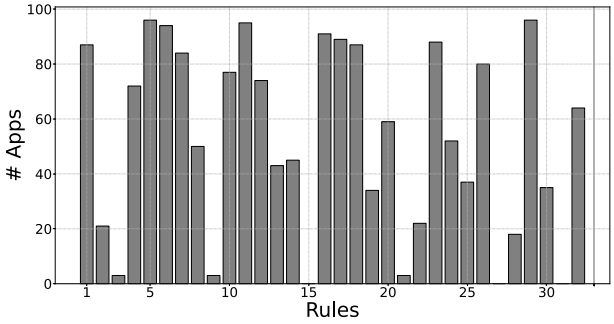

Fig. 3. Number of apps violating the rules.

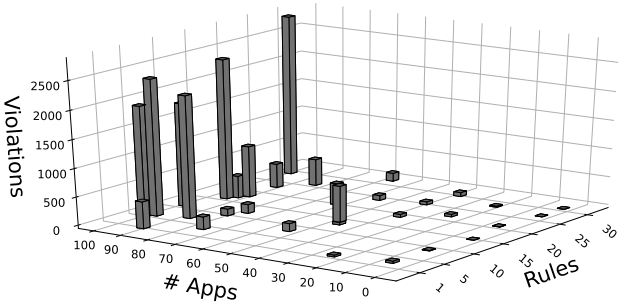

Fig. 4. Distribution of the number of violations of a rule for an app.

of the considered apps. Finally, in Section 6.7, we compare SPECK with existing static analysis tools.

6.1 Experimental Setup

To perform our experiments, we downloaded the top 100 popular apps from the Play Store according to AppBrain [71] (the full list of apps is available in Appendix C), we decompiled them through JADX and we launched SPECK against them. We executed all the experiments on an Amazon Web Service.

6.2 RQ1: What is the occurrence of vulnerabilities in Android apps?

To answer the first research question, we measured the number of apps violating a rule and the number of times the apps violate it. As shown in Fig. 3, we found that 17 rules are violated by more than the 50% of the apps, while other rules (i.e., Rule 3, Rule 9, Rule 15, Rule 21, Rule 27, Rule 28 and Rule 31) have been violated by zero or almost zero apps.

As shown in Fig. 4 (raw data is available in Table 2 in Appendix D), there are few rules violated a high number of times by a high number of apps. On the contrary, in terms of number of apps violating a rule, the majority of rules are equally distributed: some of them are violated by many apps, while some others by just a few of them.

6.3 RQ2: How long does it take for SPECK to generate an app vulnerability report?

The second research question can be addressed by measuring the average execution time required by a rule to analyze an app. In Fig. 5 (raw data is available in Table 3 and Table 4 in Appendix D), we plot the distribution of the average compilation times required by each rule to analyze one of the 100 popular apps. As shown in the plot, the rules have two clear different trends: either a very low (i.e., close to zero) or a significantly high processing time.

6.4 RQ3: What is the origin of the vulnerabilities in Android apps?

The third research question aims to determine whether a rule violation is embedded in custom code written by the developers or it belongs to Android/third-party libraries. To achieve such classification, we consider the vulnerability location within the app source code. More specifically, we consider a violation as introduced by the app developers, if this is contained in the `AndroidManifest.xml` file or in a Java file under the path that shares the name with the app package name. In any other case, we consider the vulnerability as belonging to the Android libraries or to third-party components. Fig. 6 shows that most violations are not introduced by developers custom code, but by third-party libraries, which developers rely on to enhance their apps with new functionalities.

6.5 RQ4: Which developers are more prone to introduce vulnerabilities in Android apps?

We investigated which app developers are more prone to introduce vulnerabilities in their own apps and we found that they are X-Flow (developer of Happy Color™), Voodoo (developer of Fire Balls 3D, Ball Mayhem!, Hole.io, Paper.io 2, Helix Jump) and Full Fat (developer of Grass cut), as shown in Fig. 7.

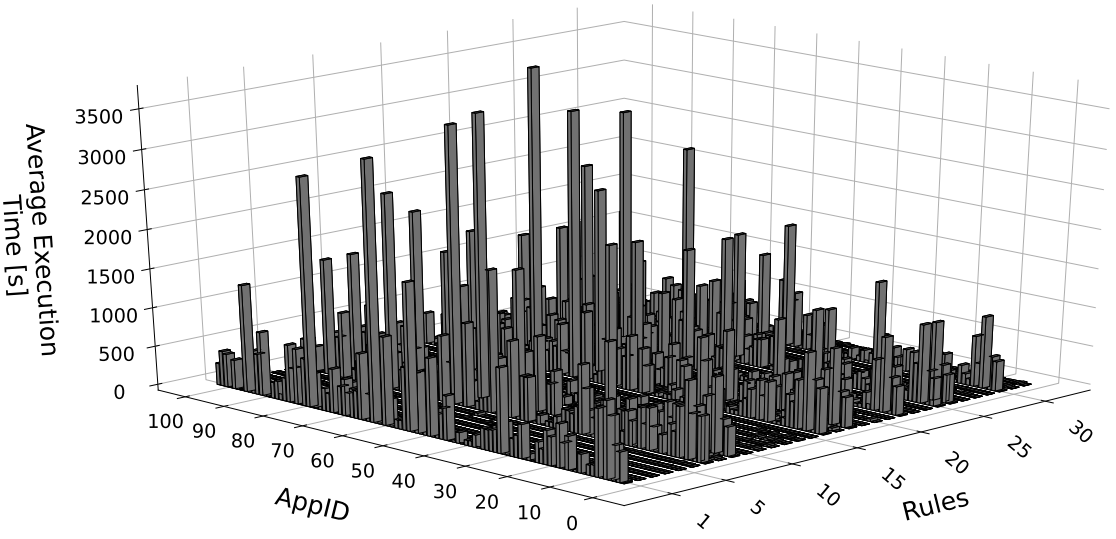

Fig. 5. Average execution time required by each rule to analyze an app.

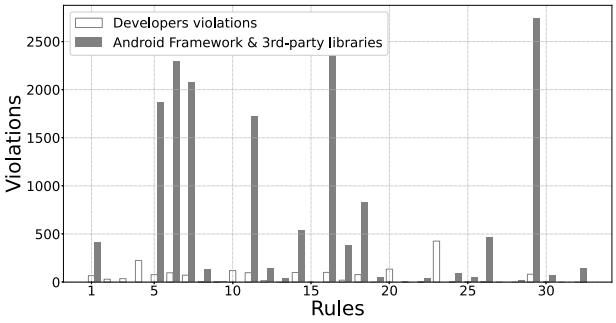

Fig. 6. Classification of the vulnerabilities introduced in Android apps according to developer code, Android framework or third-party libraries.

6.6 Rules Precision

To calculate the the statistical precision of our tool in checking the rules against the apps, we manually double-checked the correctness of the vulnerability report generated by SPECK for 10 apps out the 100 ones used in the evaluation (i.e., Instagram, Spotify, Wish, Idle Supermarket Tycoon, TextNow, Grass Cut, Samsung Notes, Twitter, Skype, Amazon). In particular, we retrieved the Java file and line of code associated to the identified vulnerability and manually verified whether it was correctly detected by SPECK. For each rule, we calculated the precision as follows:

$$TruePositive/(TruePositive + FalsePositive).$$

As shown in Table 3 (more details are available in Table 5 in Appendix D), 14 rules have a precision equal to 100% (i.e., Rule 2, Rule 4, Rule 9, Rule 10, Rule 11, Rule 12, Rule 13, Rule 16, Rule 19, Rule 21, Rule 23, Rule 24, Rule 30, Rule 32); 9 rules have a precision greater than 80% (i.e., Rule 5, Rule 6, Rule 7, Rule 8, Rule 14, Rule 20, Rule 25, Rule 26, Rule 29); 5 rules have a precision below 80% (i.e., Rule 1, Rule 3, Rule 17, Rule 18, Rule 22); no violation to Rule 15, Rule 27, Rule 28 and Rule 31 were detected, thus we were not able to calculate the precision.

6.7 Comparison with Previous Works

In this section, we provide the experimental comparison we performed to evaluate the SPECK tool against the existing ones. To achieve this aim, we inspected each tool to identify any match with the vulnerabilities detected by SPECK and then launched it against ten apps of our dataset (i.e., Instagram, Spotify, Wish, Idle Supermarket Tycoon, TextNow, Grass Cut, Samsung Notes, Twitter, Skype, Amazon). Table 4 shows the comparison between each tool and SPECK .

Argus-Amandroid. Argus-Amandroid is a static analysis framework based on the Java language, that generates an inter-component data flow graph with all the reachable components and performs on top of it a low- and context-sensitive data flow analysis. Argus-Amandroid can be used to search for security problems emerging from the interaction among components belonging to the same app or to different ones. We used Argus-Amandroid to find cryptographic and SSL/TLS related misuses. Concerning the cryptographic misuses, we compared the output of Argus-Amandroid with the SPECK Rule 22 and Rule 29. We found that SPECK detects fewer violations to Rule 22 and more to Rule 29 with respect to Argus-Amandroid. In the first case, the mismatch is due to the different approaches used by the two tools, while, in the second case, SPECK finds many more vulnerabilities because of the higher number of cryptographic algorithms considered by Rule 29. The SSL/TLS misuses detected by Argus-Amandroid can be compared with SPECK Rule 5. In this case, the different methodology leads to a significantly mismatch in the detection between the two tools.

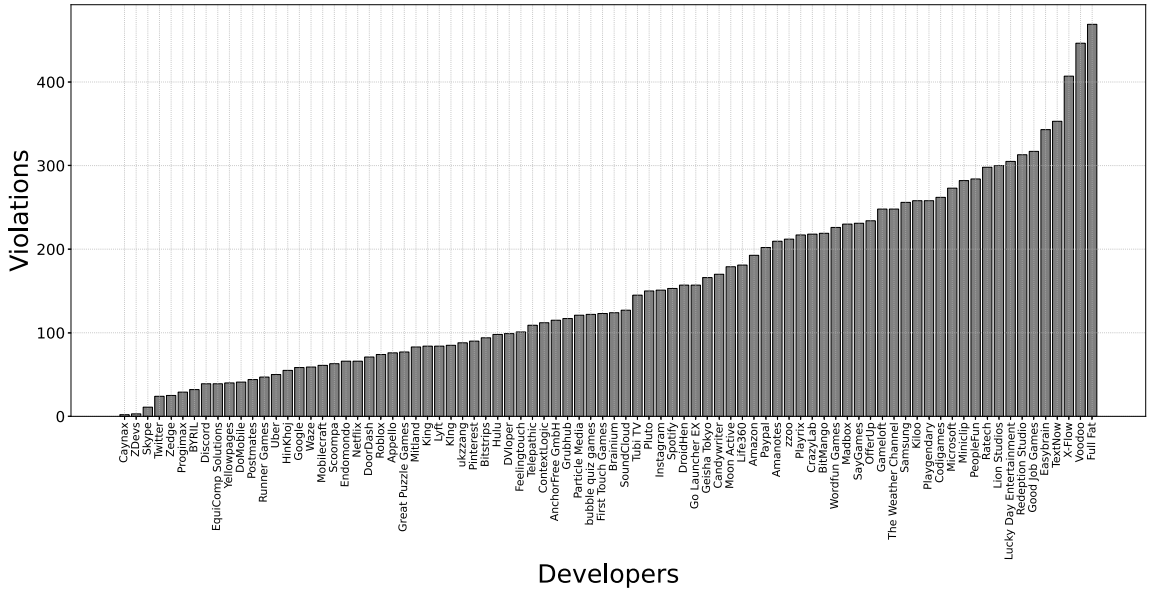

Fig. 7. Distribution of number of apps violations according to apps' developers.

TABLE 3
Rules precision, calculated by manually verifying the SPECK vulnerability report generated from the analysis of 10 apps.

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
0.78	1	0.61	1	0.86	0.98	0.98	0.94	1	1	1	1	1	0.98	-	1
R17	**R18**	**R19**	**R20**	**R21**	**R22**	**R23**	**R24**	**R25**	**R26**	**R27**	**R28**	**R29**	**R30**	**R31**	**R32**
0.66	0.64	1	0.94	1	0.50	1	1	0.85	0.81	-	1	0.96	1	-	1

TABLE 4
Comparison between the number of violations found by SPECK and the ones found by existing static analysis tools for Android apps. The numer of SPECK violations in the cells are all true positive, and are consistent with Table 5 in Appendix D. Only for the CERT TAPIOCA tool the values in the cell refer to the number of apps found vulnerable and not to the number of vulnerabilities.

	R2	R3	R5	R6	R7	R8	R13	R14	R20	R21	R22	R23	R24	R25	R26	R27	R29	R30
AMANDROID (AM)			S:197; AM: 0								S:8; AM: 19						S: 446; AM: 5	
ANDROWARN (AN)										S: 3, AN: 0								
CERT TAPIOCA (CT)			S_a: 9, CT_a: 2															
LINT (L)	S: 3 L: 0		S: 197 L: 0	S: 287 L: 0		S: 17 L: 0	S: 3 L: 0		S: 17 L: 0		S: 8 L: 0	S: 87 L: 0	S: 6 L: 0	S: 11 L: 0	S: 50 L: 0	S: 0 L: 0		S: 36 L: 0
QUARK-ENGINE (QE)													S: 6; QE:6					
RISKINDROID (R)		S: 20; R: 29																
SUPER (SU)					S: 355, SU: 8	S: 17; SU: 0		S: 17; SU: 48		S: 3; SU: 0							S: 446; SU: 32	

Androwarn. Androwarn is a static analysis tool searching for app's malicious behaviours through the inspection of specific APIs in the app's Dalvik bytecode. Among the malicious behaviours addressed by Androwarn, we found a possible comparison between the telephony services abuse and SPECK Rule 21. The mismatch is due to the different approaches used by the two tools.

CERT TAPIOCA. CERT TAPIOCA is a network-layer man-in-the-middle proxy, aimed to verify whether apps validate certificates and to inspect the HTTP/HTTPS network traffic. We identified a mapping between the first functionality of CERT TAPIOCA and SPECK Rule 5. After validating the ten apps, we found that *grasscut* and *idle supermarket tycoon* are the only two apps not validating the certificates according to CERT TAPIOCA, while SPECK finds the same vulnerability in nine apps out of ten.

Lint. Lint is the Android official scanning code tool integrated into the Android Studio IDE and aimed at identifying potential bugs related to correctness, security, performance, usability, accessibility, and internationalization. Its main purpose is to detect poorly structured code that can affect the reliability and efficiency of an Android app. In terms of detected vulnerabilities, Lint checks [72] map with several SPECK rules: "SignatureOrSystemPermissions", "ExportedContentProvider" with Rule 2; "SSLCertificateSocketFactoryCreateSocket", "SSLCertificateSocketFactoryGetInsecure", "TrustAllX509TrustManager", "UsingHttp" with Rule 5; "SetJavaScriptEnabled", "AddJavascriptInterface", "JavascriptInterface" with Rule 6; "SetWorldReadable", "SetWorldWritable", "WorldReadable-

Files", "WorldWriteableFiles" with Rule 8; "RiskyLibrary" with Rule 13; "ExportedService" with Rule 20; "SecureRandom", "TrulyRandom" with Rule 22; "UnprotectedSMSBroadcastReceiver", "ExportedReceiver", "UnsafeProtectedBroadcastReceiver" with Rule 23; "UnsafeDynamicallyLoadedCode", "UnsafeNativeCodeLocation" with Rule 24; "BadHostnameVerifier" with Rule 25; "AllowAllHostnameVerifier" with Rule 26; "HardcodedDebugMode" with Rule 27; "DeletedProvider", "DeprecatedProvider", "GetInstance" with Rule 30. However, Lint is designed to analyze source code of Android projects and not Android APK files. We decompiled the ten apps APK files and launched Lint against the decompiled source code, but Lint was not able to detect any security related issue.

Quark-Engine. Quark-Engine is a an Android malware detector, that searches for malicious activities by inspecting app's requested permissions, app's invoked native API, combination of native API, calling sequence of native API and APIs that handle the same register. Quark-Engine relies on a set of rules to inspect which APIs are invoked in the Dalvik bytecode of an app. SPECK and Quark-Engine can be compared only in terms of detection of any dynamic code loading procedure (i.e., SPECK Rule 24 and Quark-Engine Rule 21). Despite the different approach, both tools find the same vulnerabilities.

RiskInDroid. RiskInDroid calculates the risk of an Android app according to its permissions. In particular, RiskInDroid relies on a machine learning approach through which a classifier is trained over a huge dataset of apps to be able to determine whether an app is malicious or not according to its permissions. RiskInDroid identifies four types of permissions by inferring which permissions are used and which not: *declared permissions* (i.e., the ones declared in the app manifest); *exploited permissions* (i.e., the ones declared and used by the app); *ghost permissions* (i.e., the ones not declared, but used by the app); *useless permissions* (i.e., the ones declared, but never used by the app). The identification of *useless permissions* performed by RiskInDroid matches with SPECK Rule 3. The mismatch in the number of unused permissions detected by the two tools is due to the database that SPECK relies on (https://github.com/reddr/axplorer), which needed to be updated.

SUPER. SUPER analyzes APK files by applying a set of rules that search for vulnerabilities. Such rules are a combination of regular expressions and whitelisted keywords. To compare SUPER and SPECK rules, we considered only SUPER rules having a regex matching a vulnerability addressed by a SPECK rule either completely (e.g., SUPER rule "Weak Algorithms" and SPECK Rule 29) or partially (e.g., SUPER rule "WebView XSS" and SPECK Rule 7). Overall, SUPER finds fewer violations with respect to SPECK, since regular expressions are not as flexible as the algorithms used in SPECK.

7 Discussion

In this section, we discuss about the results obtained through the analysis of the Android ecosystem described in Section 6 (i.e., Section 7.1) and we illustrate the SPECK limitations (i.e., Section 7.2).

7.1 Analysis of the Android ecosystem

RQ1. To evaluate how much the 32 rules we designed are violated, we consider both the number of apps violating a specific rule and the number of violations of the apps with respect to a specific rule. Despite considering possible false positives, when SPECK finds an app violating a rule, we can claim that the app is not compliant with the guideline associated to our rule. Thus, the app might be vulnerable to the set of exploits relying on the specific vulnerability. Considering the classification of the rules according to the OWASP top ten mobile risks, the *Insecure Data Storage* category is the one with the highest number of violations, followed by *Code Tampering*, *Client Code Quality*, *Insufficient Cryptography*, *Improper Platform Usage*, *Insecure Communication* and *Insecure Authentication*. The *Extraneous Functionality* category is the one with the lowest number of violations. The most violated rules are Rule 5, Rule 6, Rule 11, Rule 29. Rule 5 refers to the use of the SSL protocol in network communications. Misuses in encrypting network communications have been already found by previous works [31], [73], that identified 1,074 apps with SSL/TLS code potentially vulnerable to Man-in-the-Middle attacks and 645 apps having `WebView` HTTPS vulnerable connections, respectively. Rule 6 refers to the malicious web code that can run in a `WebView` object, which has been already addressed by previous works [74], [75], and on the defence mechanisms that developers should introduce. The high number of violations to Rule 11 can be due to a lack of knowledge or an improper use of the associated API. Finally, Rule 29 refers to misuses of the cryptographic libraries in Android apps, which is an issue widely addressed by previous works [32], [76]. Although we found that the number of violations to the rules varies from rule to rule and some rules are not violated so often, each vulnerability is a possible attack surface and it, therefore, requires the same attention.

RQ2. Considering the time required to analyze an app, our rules have either a very low (i.e., close to zero) processing time or a significantly high one: low processing time is due to the analysis of just the AndroidManifest file, while the reason for such high processing time lies in the adoption of FlowDroid tool [5], [70], which is required by some rules (i.e., Rule 1, Rule 7, Rule 9, Rule 16, Rule 18, Rule 22, Rule 25, Rule 26, Rule 30) to perform a static analysis of the apps. In particular, FlowDroid is a static taint analysis tool that builds the call-graph of an app to search for the connection between a source and a sink object. The generation of such graph, as well as the modeling of Android lifecycles and callbacks, is a time-consuming task. The rules requiring the use of FlowDroid have a varying performance which depends on the number of violations and on the inherent complexity of the app under analysis.

RQ3. Our findings, illustrated in Section 6.4, confirm that the majority of apps vulnerabilities are introduced by third-party libraries. Previous works [77], [78], [79] already identified the possible threats introduced by third-party libraries: in [77], M. Backes et al. propose an efficient approach for detecting third-party libraries within Android apps and performed a large-scale analysis to finally identify 61 library versions affecting 296 top apps, by exposing them to crypto-analytic attacks; [78] proposes FlexDroid,

an extension of the Android permission model that allows developers to define which private information third-party libraries can have access to; in [79], M. Sun et al. developed a framework to isolate native third-party libraries from the other components of an app.

7.2 Limitations

We found some SPECK rules having a low precision mainly due to the adoption of a static analysis approach, which assumes the inspection and evaluation of an app code without its runtime execution. Thus, SPECK misses code that is dynamically loaded at runtime by an app and it cannot access runtime information or data generated during the app execution. Consequently, the rules requiring access to runtime information (e.g., Rule 11 that refers to non-sensitive app data) have a limited detection precision. Aware of this limitation, we decided that such rules print a warning, reminding the user/developer about possible malicious consequences associated to an improper usage of the specific API. In particular, the rules having lower precision are: Rule 1, Rule 3, Rule 17, Rule 18, Rule 22. Rule 1 fails when an `Intent`, first defined as implicit, then becomes explicit (e.g., through the `Intent.setComponent()` or when the `Intent.createChooser()` is called as an argument of another method). Rule 3 might find some false positives due to the outdated mappings of the Android API with the Android permissions [80], [81]. Such libraries cover up to the Android API level 25, while the latest version released is the 30^{th}. Rule 17 detects a violation whenever a `query()` method is found. However, even if present, this method might not return any result. Thus, it cannot be considered as a violation. Similarly to Rule 1, Rule 18 also fails in case a method makes an explicit `Intent` implicit. As in Rule 1 and Rule 18, Rule 22 struggles to detect if a `KeyGenerator` variable uses a `SecureRandom` object after its declaration.

Finally, FlowDroid has its own limitations: it resolves reflective calls only if their arguments are string constants; it is oblivious to native code and to multi-threading; it can not handle Android lifecycle involving new callbacks methods.

8 Conclusion

The Android OS is getting more and more complex, providing new functionalities in every new release. While Android developers have to keep up the pace with such evolution, trying to make their apps attractive for mobile users, they have also to consider the role of the attackers, that are willing to exploit the Android apps vulnerabilities. To this aim, researchers proposed several tools aimed at detecting vulnerabilities in Android apps, but most of them focus on a single class of vulnerabilities and none of them on preventing developers from introducing vulnerabilities in their own apps. Google provides a set of guidelines concerning security and privacy issues of Android apps, which, however, require a strong involvement by developers. We believe there is an urgent need to release automatic tools that can help developers with preventing the introduction of software vulnerabilities in their apps. Thus, we first analyze the Google security and privacy guidelines, currently available in a textual format, and we "translate" them into 32 rules. We, then, propose SPECK (Security and Privacy chECK of Android apps vulnerabilities), a rule-based static analysis system that automatically finds the violations to our rules. In particular, for each violated rule, SPECK shows the developer the specific line of code where the vulnerability has been detected, thus prompting him to fix the issue. We manually validated the precision of the 32 rules and, then, analyzed the Android ecosystem by launching SPECK against 100 popular apps. We found that each app has at least one violation to our rules, while more than the 50% of them violates at least 17 rules. Few rules are violated by almost all the apps (some of them even multiple times by the same app). The majority of violations (90.13%) are located in external libraries. The developers more prone to errors are X-Flow (developer of Happy Color™), Voodoo (developer of Fire Balls 3D, Ball Mayhem!, Hole.io, Paper.io 2, Helix Jump) and Full Fat (developer of Grass cut).

9 Acknowledgement

We would like to thank Julien Branlant, for his contribution in the design and development of the 32 rules and of the SPECK system, as well as Michele Agnello and Alberto Molon, for their help in improving the experimental evaluation of SPECK.

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the LOCARD project (Grant Agreement no. 832735).

References

[1] "Mobile Operating System Market Share Worldwide," 2020, last access: June 20, 2022. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 121–136.

[3] "Google Play Protect," 2020, last access: June 20, 2022. [Online]. Available: https://developer.android.com/training/articles/security-tips

[4] "Improve your code with lint checks," 2017, last access: June 20, 2022. [Online]. Available: https://developer.android.com/studio/write/lint

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "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*, ser. PLDI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 259–269. [Online]. Available: https://doi.org/10.1145/2594291.2594299

[6] "Droidsafe," 2016, last access: June 20, 2022. [Online]. Available: https://github.com/MIT-PAC/droidsafe-src

[7] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe Exposure Analysis of Mobile In-App Advertisements," in *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WISEC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 101–112. [Online]. Available: https://doi.org/10.1145/2185448.2185464

[8] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 229–240. [Online]. Available: https://doi.org/10.1145/2382196.2382223

[9] "Blueseal," 2014, last access: June 20, 2022. [Online]. Available: http://blueseal.cse.buffalo.edu/

[10] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android Taint Flow Analysis for App Sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, ser. SOAP '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6. [Online]. Available: https://doi.org/10.1145/2614628.2614633

[11] "Anadroid," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/RRua/AnaDroid

[12] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The Impact of Vendor Customizations on Android Security," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 623–634. [Online]. Available: https://doi.org/10.1145/2508859.2516728

[13] K. Lu, Z. Li, V. P. Kemerlis, Z. Wu, L. Lu, C. Zheng, Z. Qian, W. Lee, and G. Jiang, "Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[14] X. Xiao, N. Tillmann, M. Fahndrich, J. De Halleux, and M. Moskal, "User-Aware Privacy Control via Extended Static-Information-Flow Analysis," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: Association for Computing Machinery, 2012, p. 80–89. [Online]. Available: https://doi.org/10.1145/2351676.2351689

[15] K. O. Elish, D. Yao, and B. Ryder, "User-Centric Dependence Analysis For Identifying Malicious Mobile Apps," in *Proceedings of the Workshop on Mobile Security Technologies 2012 (MoST'12)*, 2012.

[16] X. Chen and S. Zhu, "DroidJust: Automated Functionality-Aware Privacy Leakage Analysis for Android Applications," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2766498.2766507

[17] J. Kim, Y. Yoon, and K. Yi, "SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications," in *Proceedings of the Workshop on Mobile Security Technologies (MoST'12)*, 2012.

[18] X. Cui, J. Wang, L. C. K. Hui, Z. Xie, T. Zeng, and S. M. Yiu, "WeChecker: Efficient and Precise Detection of Privilege Escalation Vulnerabilities in Android Apps," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2766498.2766509

[19] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1043–1054. [Online]. Available: https://doi.org/10.1145/2508859.2516676

[20] J. P. Achara, M. Cunche, V. Roca, and A. Francillon, "Short Paper: WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission," in *7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, Oxford, United Kingdom, Jul. 2014. [Online]. Available: https://hal.inria.fr/hal-00997716

[21] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 576–587. [Online]. Available: https://doi.org/10.1145/2635868.2635869

[22] M. A. El-Zawawy, E. Losiouk, and M. Conti, "Do not let Next-Intent Vulnerability be your next nightmare: type system-based approach to detect it in Android apps," *International Journal of Information Security*, Mar. 2020. [Online]. Available: http://link.springer.com/10.1007/s10207-020-00491-x

[23] ——, "Vulnerabilities in android webview objects: Still not the end!" *Computers Security*, vol. 109, p. 102395, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002194

[24] A. Pham, I. Dacosta, E. Losiouk, J. Stephan, K. Huguenin, and J.-P. Hubaux, "Hidemyapp: Hiding the presence of sensitive apps on android," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 711–728. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/pham

[25] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 627–638. [Online]. Available: https://doi.org/10.1145/2046707.2046779

[26] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 217–228. [Online]. Available: https://doi.org/10.1145/2382196.2382222

[27] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus, "Automatically Securing Permission-Based Software by Reducing the Attack Surface: An Application to Android," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: Association for Computing Machinery, 2012, p. 274–277. [Online]. Available: https://doi.org/10.1145/2351676.2351722

[28] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon, "Effective Inter-Component Communication Mapping in Android: An Essential Step Towards Holistic Security Analysis," in *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 543–558. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/octeau

[29] D. Gallingani and R. Gjomemo, "Static Detection and Automatic Exploitation of Intent Message Vulnerabilities in Android Applications," in *Proceedings of the Workshop on Mobile Security Technologies (MoST'12)*, 2012.

[30] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," ser. MobiSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 239–252. [Online]. Available: https://doi.org/10.1145/1999995.2000018

[31] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 50–61. [Online]. Available: https://doi.org/10.1145/2382196.2382205

[32] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–84. [Online]. Available: https://doi.org/10.1145/2508859.2516693

[33] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection," *Trans. Info. For. Sec.*, vol. 13, no. 5, p. 1286–1300, May 2018.

[34] M. Sun, M. Li, and J. C. S. Lui, "DroidEagle: Seamless Detection of Visually Similar Android Apps," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2766498.2766508

[35] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "ViewDroid: Towards Obfuscation-Resilient Mobile Application Repackaging Detection," in *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, ser. WiSec '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 25–36. [Online]. Available: https://doi.org/10.1145/2627393.2627395

[36] "Argus static analysis framework," 2018, last access: June 20, 2022. [Online]. Available: http://pag.arguslab.org/argus-saf

[37] "Androwarn," 2019, last access: June 20, 2022. [Online]. Available: https://github.com/maaaaz/androwarn

[38] "Apkanalyser," 2013, last access: June 20, 2022. [Online]. Available: https://github.com/sonyxperiadev/ApkAnalyser

[39] "Apkinspector," 2013, last access: June 20, 2022. [Online]. Available: https://github.com/honeynet/apkinspector/

[40] "Apkleaks," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/dwisiswant0/apkleaks

[41] "apkx - android apk decompilation for the lazy," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/b-mueller/apkx

[42] F. Shen, N. Vishnubhotla, C. Todarka, M. Arora, B. Dhandapani, E. J. Lehner, S. Y. Ko, and L. Ziarek, "Information Flows as a Permission Mechanism," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 515–526. [Online]. Available: https://doi.org/10.1145/2642937.2643018

[43] "Cfgscandroid," 2015, last access: June 20, 2022. [Online]. Available: https://github.com/TACIXAT/CFGScanDroid

[44] "Classyshark," 2020, last access: June 20, 2022. [Online]. Available: https://github.com/google/android-classyshark

[45] "Condroid," 2016, last access: June 20, 2022. [Online]. Available: https://github.com/JulianSchuette/ConDroid

[46] "Droidlegacy," 2016, last access: June 20, 2022. [Online]. Available: https://bitbucket.org/srl/droidlegacy/src/master/

[47] "Droidra," 2017, last access: June 20, 2022. [Online]. Available: https://github.com/serval-snt-uni-lu/DroidRA

[48] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information Flow Analysis of Android Applications in DroidSafe," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. [Online]. Available: https://www.ndss-symposium.org/ndss2015/information-flow-analysis-android-applications-droidsafe

[49] "Jaadas: Joint advanced application defect assessment for android application," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/flankerhqd/JAADAS

[50] "maldrolyzer," 2015, last access: June 20, 2022. [Online]. Available: https://github.com/maldroid/maldrolyzer

[51] "Quark-engine," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/quark-engine/quark-engine

[52] "Riskindroid," 2020, last access: June 20, 2022. [Online]. Available: https://github.com/ClaudiuGeorgiu/RiskInDroid

[53] A. Merlo and G. C. Georgiu, "Riskindroid: Machine learning-based risk analysis on android," in *ICT Systems Security and Privacy Protection*, S. De Capitani di Vimercati and F. Martinelli, Eds. Cham: Springer International Publishing, 2017, pp. 538–552.

[54] "Smali-cfgs," 2014, last access: June 20, 2022. [Online]. Available: https://github.com/EugenioDelfa/Smali-CFGs

[55] "Smalisca: Static code analysis for smali," 2017, last access: June 20, 2022. [Online]. Available: https://github.com/dorneanu/smalisca

[56] "Sparta! static program analysis for reliable trusted apps," 2016, last access: June 20, 2022. [Online]. Available: https://types.cs.washington.edu/sparta/

[57] "Stacoan," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/vincentcox/StaCoAn

[58] "Super android analyzer," 2018, last access: June 20, 2022. [Online]. Available: https://github.com/SUPERAndroidAnalyzer/super

[59] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–84. [Online]. Available: https://doi.org/10.1145/2508859.2516693

[60] J. Tang, J. Li, R. Li, H. Han, X. Gu, Z. Xu, and P. Nicopolitidis, "Ssldetecter: Detecting ssl security vulnerabilities of android applications based on a novel automatic traversal method," *Sec. and Commun. Netw.*, vol. 2019, jan 2019. [Online]. Available: https://doi.org/10.1155/2019/7193684

[61] S. Salva and S. Zafimiharisoa, "APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities," *International Journal on Software Tools for Technology Transfer*, vol. 17, pp. 201–, 02 2015.

[62] P. Gadient, M. Ghafari, P. Frischknecht, and O. Nierstrasz, "Security code smells in android icc," *Empir Software Eng*, vol. 24, p. 3046–3076, 2019.

[63] J. Gajrani, M. Tripathi, V. Laxmi, G. Somani, A. Zemmari, and M. S. Gaur, "Vulvet: Vetting of vulnerabilities in android apps to thwart exploitation," *Digital Threats*, vol. 1, no. 2, may 2020. [Online]. Available: https://doi.org/10.1145/3376121

[64] H. Shahriar and H. M. Haddad, "Content provider leakage vulnerability detection in android applications," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 359–366. [Online]. Available: https://doi.org/10.1145/2659651.2659716

[65] D. Bassolé, Y. Traoré, G. Koala, F. Tchakounté, and O. Sié, "Detection of vulnerabilities related to permissions requests for android apps using machine learning techniques," in *SoCPaR*, 2020.

[66] B. F. Demissie, M. Ceccato, and L. K. Shar, "Security analysis of permission re-delegation vulnerabilities in Android apps," 3 2021. [Online]. Available: https://researchdata.smu.edu.sg/articles/journal_contribution/Security_analysis_of_permission_re-delegation_vulnerabilities_in_Android_apps/14236268

[67] D. Wu and R. K. C. Chang, "Analyzing android browser apps for file: // vulnerabilities," in *ISC*, 2014.

[68] "Cert tapioca," 2021, last access: June 20, 2022. [Online]. Available: https://github.com/CERTCC/tapioca

[69] "Top 10 Mobile Risks," 2020, last access: June 20, 2022. [Online]. Available: https://owasp.org/www-project-mobile-top-10/

[70] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *SIGPLAN Not.*, vol. 49, no. 6, p. 259–269, Jun. 2014. [Online]. Available: https://doi.org/10.1145/2666356.2594299

[71] "AppBrain," 2020, last access: June 20, 2022. [Online]. Available: https://www.appbrain.com/stats/google-play-rankings

[72] "Android lint checks," 2021, last access: June 20, 2022. [Online]. Available: http://tools.android.com/tips/lint-checks

[73] C. Zuo, J. Wu, and S. Guo, "Automatically Detecting SSL Error-Handling Vulnerabilities in Hybrid Mobile Web Apps," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 591–596. [Online]. Available: https://doi.org/10.1145/2714576.2714583

[74] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G. N. Peri, "Code Injection Attacks on HTML5-Based Mobile Apps: Characterization, Detection and Mitigation," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 66–77. [Online]. Available: https://doi.org/10.1145/2660267.2660275

[75] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A Large-Scale Study of Mobile Web App Security," in *Proceedings of the Workshop on Mobile Security Technologies (MoST'12)*, 2015.

[76] I. Muslukhov, Y. Boshmaf, and K. Beznosov, "Source attribution of cryptographic api misuse in android applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 133–146. [Online]. Available: https://doi.org/10.1145/3196494.3196538

[77] M. Backes, S. Bugiel, and E. Derr, "Reliable Third-Party Library Detection in Android and Its Security Applications," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 356–367. [Online]. Available: https://doi.org/10.1145/2976749.2978333

[78] J. Seo, D. Kim, D. Cho, I. Shin, and T. Kim, "FLEXDROID: Enforcing In-App Privilege Separation in Android," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/flexdroid-enforcing-in-app-privilege-separation-android.pdf

[79] M. Sun and G. Tan, "NativeGuard: Protecting Android Applications from Third-Party Native Libraries," in *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 165–176. [Online]. Available: https://doi.org/10.1145/2627393.2627396

[80] "axplorer," 2020, last access: June 20, 2022. [Online]. Available: https://github.com/reddr/axplorer

[81] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, p. 1101–1118.

Mauro Conti is Full Professor at the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor the University of Padua, where he became Associate Professor in 2015, and Full Professor in 2018. He has been Visiting Researcher at GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest is in the area of Security and Privacy. In this area, he published more than 400 papers in topmost international peer-reviewed journals and conferences. He is Area Editor-in-Chief for IEEE Communications Surveys Tutorials, and has been Associate Editor for several journals, including IEEE Communications Surveys Tutorials, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Network and Service Management. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, and General Chair for SecureComm 2012, SACMAT 2013, NSS 2021 and ACNS 2022. He is Senior Member of the IEEE and ACM. He is a member of the Blockchain Expert Panel of the Italian Government. He is Fellow of the Young Academy of Europe. From 2020, he is Head of Studies of the Master Degree in Cybersecurity at University of Padua.

Eleonora Losiouk is an Assistant Professor at the University of Padua (Italy), working in the SPRITZ Group led by Prof. Mauro Conti. In 2018, she obtained her Ph.D. in Bioengineering and Bioinformatics from the University of Pavia (Italy). She has been a Visiting Fellow at EPFL in 2017. In 2020, she received the Seal of Excellence for her Marie Skłodowska-Curie individual project proposal and was awarded a Fulbright Fellowship for visiting ICSI, Berkeley (USA). Her main research interests regard the security and privacy evaluation of the Android Operating System.

Roberto Rossini is a MSc in Computer Engineering. He studied at the Università degli Studi di Padova for both his BSc in Information Engineering and his MSc in Computer Engineering. He was an exchange student at the National Chiao Tung University in Hsinchu, Taiwan and at the Eidgenössische Technische Hochschule in Zürich, Switzerland. He is passionate in cybersecurity, in particular in the field of mobile and web applications.

APPENDIX A
RULES

A.1 Rule 2 - Content provider access control

A.1.1 Apply signature-based permissions

Google guideline. *When sharing data between two apps that you control or own, use signature-based permissions. These permissions don't require user confirmation and instead check that the apps accessing the data are signed using the same signing key. Therefore, these permissions offer a more streamlined, secure user experience.*

```
<manifest xmlns:android="http://schemas.android.com
   /apk/res/android" package="com.example.myapp">
  <permission
    android:name="my_custom_permission_name"
    android:protectionLevel="signature" />
```

Listing 1. Apply signature-based permissions

A.1.2 Disallow access to your app's content providers

Google guideline. *Unless you intend to send data from your app to a different app that you don't own, you should explicitly disallow other developers' apps from accessing the* `ContentProvider` *objects that your app contains. This setting is particularly important if your app can be installed on devices running Android 4.1.1 (API level 16) or lower, as the* `android:exported` *attribute of the* `<provider>` *element is* `true` *by default on those versions of Android.*

```
<manifest xmlns:android="http://schemas.android.com
   /apk/res/android" package="com.example.myapp">
  <application ... >
    <provider
      android:name="android.support.v4.content.
   FileProvider"
      android:authorities="com.example.myapp.
   fileprovider"
      ...
      android:exported="false">
      <!-- Place child elements of <provider> here.
    -->
    </provider>
    ...
  </application>
</manifest>
```

Listing 2. Disallow access to your app's content providers

Pseudo-code. The rule we formalized in shown in Algorithm 2.

Rule 2: Content provider access control

```
begin
    providers ← getContentProviderObjs()
    apiLevel ← getMinApiLevel()
    foreach provider in providers do
        if isExported(provider, apiLevel) then
            if not useCustomPermission(provider)
             then
                Rule 2 is not respected.
            end
        end
    end
end
```

Attack. If a `ContentProvider` object is *exported* and not permission-protected, any other app on the same device can interact with it, by launching SQL injection attacks, reading or modifying its data.

A.2 Rule 3 - Provide the right permissions

Google guideline. *Your app should request only the minimum number of permissions necessary to function properly. When possible, your app should relinquish some of these permissions when they're no longer needed.*

Pseudo-code. The rule we formalized in shown in Algorithm 3. For the mapping between the Android API and the Android permissions, we used the online database (https://github.com/reddr/axplorer).

Rule 3: Provide the right permissions

```
begin
    permissions ← getAllPermissions()
    foreach perm in permissions do
        sdkFuncs ← getSdkFuncs(perm)
        uriContProvs ← getURIContProvs(perm)
        names ← sdkFuncs + uriContProvs
        if not allInJavaCode(names) then
            Rule 3 is not respected.
        end
    end
end
```

Attack. To access protected resources on a mobile device, an app has to declare the associated permissions. Any code running within the same UID has access to the same set of protected resources, defined according to the permissions declared by the app. This can also happen for third-party libraries, which an app might include to have additional features. The higher the number of permissions declared by an app, the higher the risk for the whole mobile device to get attacked malicious code running within the same UID of that app.

A.3 Rule 4 - Use intents to defer permissions

Google guideline. *Whenever possible, don't add a permission to your app to complete an action that could be completed in another app. Instead, use an intent to defer the request to a different app that already has the necessary permission.*
The following example shows how to use an intent to direct users to a contacts app instead of requesting the `READ_CONTACTS` *and* `WRITE_CONTACTS` *permissions:*

```
// Delegates the responsibility of creating the
// contact to a contacts app, which has already
// been granted the appropriate WRITE_CONTACTS
// permission.
Intent insertContactIntent = new  Intent(Intent.
   ACTION_INSERT);
insertContactIntent.setType(ContactsContract.
   Contacts.CONTENT_TYPE);

// Make sure that the user has a contacts app
// installed on their device.
if  (insertContactIntent.resolveActivity(
   getPackageManager()) != null) {
  startActivity(insertContactIntent);
}
```

Listing 3. Use intents to defer permissions

In addition, if your app needs to perform file-based I/O – such as accessing storage or choosing a file – it doesn't need special permissions because the system can complete the operations on your app's behalf. Better still, after a user selects content at a particular URI, the calling app gets granted permission to the selected resource.

Pseudo-code. The rule we formalized is shown in Algorithm 4.

Rule 4: Use intents to defer permissions

```
begin
    permissions ← getAllPermissions()
    foreach perm in permissions do
        if perm in blacklist then
            Rule 4 is not respected.
        end
    end
end
```

Attack. The attack that can exploit the vulnerability detected by Rule 4 is the same as the one for Rule 3 in Section A.2.

A.4 Rule 7 - Use WebView objects carefully

Google guideline. *Whenever possible, load only whitelisted content in `WebView` objects. In other words, the `WebView` objects in your app shouldn't allow users to navigate to sites that are outside of your control.*

Pseudo-code. The rule we formalized is shown in Algorithm 7.

Rule 7: Use WebView objects carefully

```
begin
    webViews ← getAllWebViewVars()
     whitelistedViews ← getSetWebViewClient()
     foreach webView in webViews do
        respected ← False
        foreach view in whitelistedViews do
            if webView = view then
                respected ← True
                break
            end
        end
        if not respected then
            Rule 7 is not respected.
        end
    end
    foreach view in whitelistedViews do
        if not isOverridingUrlLoading(view) then
            Rule 7 is not respected.
        end
    end
end
```

Attack. `WebView` objects are responsible for rendering the web code either belonging to external resources (e.g., a website) or saved in an app. If a `WebView` object loads any website and does not refer to a specific whitelist, an attacker might make the `WebView` object load a malicious website, which has JavaScript code running on the client side and able to steal sensitive information (e.g., cookies).

A.5 Rule 8 - Store private data within internal storage

Google guideline. *Store all private user data within the device's internal storage, which is sandboxed per app. Your app doesn't need to request permission to view these files, and other apps cannot access the files. As an added security measure, when the user uninstalls an app, the device deletes all files that the app saved within internal storage.*
The following code snippet demonstrates one way to write data to storage:

```
// Creates a file with this name, or replaces an
// existing file that has the same name. Note that
// the file name cannot contain path separators.
final String FILE_NAME = "sensistive_info.txt";
String fileContents = "This is some top-secret
    information!";

FileOutputStream fos = openFileOutput(FILE_NAME,
    Context.MODE_PRIVATE);
fos.write(fileContents.getBytes());
fos.close();
```

Listing 4. Write data to the internal storage

The following code snippet shows the inverse operation, reading data from storage:

```
// The file name cannot contain path separators.
final String FILE_NAME = "sensitive_info.txt";
FileInputStream fis = openFileInput(FILE_NAME);

// available() determines the approximate number of
// bytes that can be read without blocking.
int bytesAvailable = fis.available();
StringBuilder topSecretFileContents = new
    StringBuilder(bytesAvailable);

// Make sure that read() returns a number of bytes
// that is equal to the file's size.
byte[] fileBuffer = new byte[bytesAvailable];
while (fis.read(fileBuffer) != -1) {
  topSecretFileContents.append(fileBuffer);
}
```

Listing 5. Read data from the internal storage

Pseudo-code. The rule we formalized is shown in Algorithm 8.

Rule 8: Store private data within internal storage

```
begin
    s1 ← "openFileOutput"
    s2 ← "MODE_PRIVATE"
    methods ← getAllCalledMethods()
    foreach method in methods do
        if method = s1 then
            mode ← getModeArg(method)
            if not mode = s2 then
                Rule 8 is not respected.
            end
        end
    end
end
```

Attack. An attacker can read and pollute data since they are not stored in the app private internal storage. Moreover, through a a *Man-in-the-Disk* attack, an attacker can intercept and potentially alter data while they are extracted by an app from the external storage.

A.6 Rule 9 - Share data securely across apps

Google guideline. *Follow these best practices in order to share your app's content with other apps in a more secure manner:*

- *Enforce read-only or write-only permissions as needed.*
- *Provide clients one-time access to data by using the* `FLAG_GRANT_READ_URI_PERMISSION` *and* `FLAG_GRANT_WRITE_URI_PERMISSION` *flags.*
- *When sharing data, use "content://" URIs, not "file://" URIs. Instances of* `FileProvider` *do this for you.*

The following code snippet shows how to use URI permission grant flags and content provider permissions to display an app's PDF file in a separate PDF Viewer app:

```
// Create an Intent to launch a PDF viewer for a
// file owned by this app.
Intent viewPdfIntent = new Intent(Intent.
    ACTION_VIEW);
viewPdfIntent.setData(Uri.parse("content://com.
    example/personal-info.pdf"));

// This flag gives the started app read access to
// the file.
viewPdfIntent.addFlags(Intent.
    FLAG_GRANT_READ_URI_PERMISSION);

// Make sure that the user has a PDF viewer app
// installed on their device.
if (viewPdfIntent.resolveActivity(
  getPackageManager()) != null) {
  startActivity(viewPdfIntent);
}
```

Listing 6. Share data securely across apps

Pseudo-code. The rule we formalized is shown in Algorithm 9.

Rule 9: Share data securely across apps

```
begin
    str ← "file:/"
    arr ← ["FLAG_GRANT_READ_URI_PERMIS-
    SION", "FLAG_GRANT_WRITE_URI_PERMIS-
    SION"]
    setDataIntents ← getSetDataIntents()
    foreach intent in setDataIntents do
        uriScheme ← getURIScheme(intent)
        if uriScheme = str then
            Rule 9 is not respected.
        end
        flagArg ← getAddFlagsArg(intent)
        if not flagArg in arr then
            Rule 9 is not respected.
        end
    end
end
```

Attack. URI permissions can be used to grant other apps access to specific URIs. These permissions are temporary and expire automatically when the receiving app's task stack is finished. However, if the URI to access a file is declared as `"file://"`, the file system permissions are changed and they allow anyone to access the file.

A.7 Rule 10 - Use scoped directory access

Google guideline. *If your app needs to access only a specific directory within the device's external storage, you can use scoped directory access to limit your app's access to a device's external storage accordingly. As a convenience to users, your app should save the directory access URI so that users don't need to approve access to the directory every time your app attempts to access it.*
***Note**: if you use scoped directory access with a particular directory in external storage, know that the user might eject the media containing this storage while your app is running. You should include logic to gracefully handle the change to the* `Environment.getExternalStorageState()` *return value that this user behaviour causes.*
The following code snippet uses scoped directory access with the pictures directory within a device's primary shared storage:

```
private static final int
    PICTURES_DIR_ACCESS_REQUEST_CODE = 42;

private void accessExternalPicturesDirectory() {
  StorageManger sm = (StorageManager)
   getSystemService(Context.STORAGE_SERVICE);
  StorageVolume = sm.getPrimaryStorageVolume();
  Intent intent = volume.createAccessIntent(
   Environment.DIRECTORY_PICTURES);
  startActivityForResult(intent,
   PICTURES_DIR_ACCESS_REQUEST_CODE);
}

...

@Override
public void onActivityResult(int requestCode, int
    resultCode, Intent resultData) {
  if (requestCode==PICTURES_DIR_ACCESS_REQUEST_CODE
     && resultCode == Activity.RESULT_OK) {

    // User approved access to scoped directory in
    // your app
    if (resultData != null) {
      Uri picturesDirUri = resultData.getData();

      // Save user's approval for accessing this
      // directory in your app
      ContentResolver myContentResolver =
   getContentResolver();
      myContentResolver.
   takePersistableUriPermission(picturesDirUri,
   Intent.FLAG_GRANT_READ_URI_PERMISSION);
    }
  }
}
```

Listing 7. Use scoped directory access

***Warning**: don't pass null into* `createAccessIntent()` *unnecessarily because this grants your app access to the entire volume that* `StorageManager` *finds for your app.*

Pseudo-code. The rule we formalized is shown in Algorithm 10.

Rule 10: Use scoped directory access

```
begin
    arr ← ["READ_EXTERNAL_STORAGE",
     "WRITE_EXTERNAL_STORAGE"]
    permissions ← getAllPermissions()
    foreach perm in permissions do
        if perm in arr then
            Rule 10 is not respected.
        end
    end
end
```

Attack. As for Rule 3 in Section A.2, according to which an app should declare the minimum number of permissions, Rule 10 aims to prevent any malicious code running within the same UID of the app from having access to the whole external storage. Thus, if Rule 10 is not respected and the app has access to the external storage, any malicious code running inside it can not only compromise the app files, but also the ones belonging to other apps.

A.8 Rule 12 - Use SharedPreferences in private mode

Google guideline. *When using `getSharedPreferences` to create or access your app's `SharedPreferences` objects, use `MODE_PRIVATE`. That way, only your app can access the information within the shared preferences file.*
If you want to share data across apps, don't use `SharedPreferences` objects. Instead, you should follow the necessary steps to share data securely across apps.

Pseudo-code. The Rule 12 pseudo-code is shown in Algorithm 12.

Rule 12: Use SharedPreferences in private mode

```
begin
    s1 ← "getSharedPreferences"
    s2 ← "MODE_PRIVATE"
    methods ← getAllCalledMethods()
    foreach method in methods do
        if method = s1 then
            mode ← getModeArg(method)
            if not mode = s2 then
                Rule 12 is not respected.
            end
        end
    end
end
```

Attack. If an app accesses to its `SharedPreferences` without the `MODE_PRIVATE`, a malicious app on the same device can access the same and read/modify the stored information.

A.9 Rule 13 - Keep services and dependencies up-to-date

Google guideline. *Most apps use external libraries and device system information to complete specialized tasks. By keeping your app's dependencies up to date, you make these points of communication more secure.*

A.9.1 Check the Google Play services security provider

***Note**: this section applies only to apps targeting devices that have Google Play services installed.*
If your app uses Google Play services, make sure that it's updated on the device where your app is installed. This check should be done asynchronously, off of the UI thread. If the device isn't up-to-date, your app should trigger an authorization error.
To determine whether Google Play services is up to date on the device where your app is installed, follow the steps in the guide for "Updating Your Security Provider to Protect Against SSL Exploits"[1].

1. https://developer.android.com/training/articles/security-gms-provider

Pseudo-code. The rule we formalized is shown in Algorithm 13.

Rule 13: Keep services and dependencies up-to-date

```
begin
    s1 ← "ProviderInstaller.installIfNeeded"
    s2 ← "ProviderInstaller.installIfNeededAsync"
    if playServicesDirExists() then
        respected ← False
        javaCode ← getAllJavaCode()
        foreach word in javaCode do
            if word in [s1, s2] then
                respected ← True
            end
        end
        if not respected then
            Rule 13 is not respected.
        end
    end
end
```

Attack. Not keeping Google Play services or third-party libraries up-to-date would let an Android application vulnerable to some known vulnerabilities. An attacker might exploit these vulnerabilities, which have been already identified and published.

A.10 Rule 14 - Check validity of data

Google guideline. *If your app uses data from external storage, make sure that the contents of the data haven't been corrupted or modified. Your app should also include logic to handle files that are no longer in a stable format.*
An example of a hash verifier appears in the following code snippet:

```
Executor threadPoolExecutor = Executors.
    newFixedThreadPool(4);
private interface HashCallback {
  void onHashCalculated(@Nullable String hash);
}

boolean hashRunning = calculateHash(inputStream,
    threadPoolExecutor, hash -> {
  if (Objects.equals(hash, expectedHash)) {
    // Work with the content.
  }
});

if (!hashRunning) {
  // There was an error setting up the hash
  // function.
}

private boolean calculateHash(@NonNull InputStream
    stream, @NonNull Executor executor, @NonNull
    HashCallback hashCallback) {
  final MessageDigest digest;
  try {
    digest = MessageDigest.getInstance("SHA-512");
  } catch (NoSuchAlgorithmException nsa) {
    return false;
  }

  // Calculating the hash code can take quite a bit
  // of time, so it shouldn't be done on the main
  // thread.
```

```
  executor.execute(() -> {
    String hash;
    try (DigestInputStream digestStream =
      new DigestInputStream(stream, digest)) {
        while (digestStream.read() != -1) {
          // The DigestInputStream does the work;
          // nothing for us to do.
        }
        StringBuilder builder=new StringBuilder();
        for (byte aByte : digest.digest()) {
          builder.append(String.format("%02x",
   aByte)).append(':');
        }
        hash = builder.substring(0,
        builder.length() - 1);
    } catch (IOException e) {
      hash = null;
    }

    final String calculatedHash = hash;
    runOnUiThread(() -> hashCallback.
   onHashCalculated(calculatedHash));
  });
  return true;
}
```

Listing 8. Check validity of data

Pseudo-code. The rule we formalized is shown in Algorithm 14.

Rule 14: Check validity of data

```
begin
    str ← "READ_EXTERNAL_STORAGE"
    permissions ← getAllPermissions()
    foreach perm in permissions do
        if perm = str then
            vars ← getAllFileInputVars()
            foreach var in vars do
                if not checkValidity(var) then
                    Rule 14 is not respected.
                end
            end
        end
    end
end
```

Attack. If an app does not check the validity of the data stored on the external storage, it might not rely that some data could have been tampered with by a malicious app on the same device.

A.11 Rule 15 - Create permissions

Google guideline. *Generally, you should strive to define as few permissions as possible while satisfying your security requirements. Creating a new permission is relatively uncommon for most applications, because the system-defined permissions cover many situations. Where appropriate, perform access checks using existing permissions.*

If you must create a new permission, consider whether you can accomplish your task with a signature protection level. Signature permissions are transparent to the user and allow access only by applications signed by the same developer as the application performing the permission check. If the new permission is still required, it's declared in the app manifest using the `<permission>` *element. Apps that wish to use the new permission can reference it by each adding a* `<uses-permission>` *element in their respective manifest files. You can also add permissions dynamically by using the* `addPermission()` *method.*

If you create a permission with the dangerous protection level, there are a number of complexities that you need to consider:

- *The permission must have a string that concisely expresses to a user the security decision they are required to make.*
- *The permission string must be localized to many different languages.*
- *Users may choose not to install an application because a permission is confusing or perceived as risky.*
- *Applications may request the permission when the creator of the permission has not been installed.*

Each of these poses a significant nontechnical challenge for you as the developer while also confusing your users, which is why we discourages the use of the dangerous permission level.

Pseudo-code. The rule we formalized is shown in Algorithm 15.

Rule 15: Create permissions

```
begin
    str ← "dangerous"
    permissions ← getCustomPermissions()
    foreach perm in permissions do
        if getPermProtectLevel(perm) = str then
            Rule 15 is not respected.
        end
    end
end
```

Attack. Defining new permissions without the signature protection level might lead to a lack of access control to protected resources. Any malicious app can declare the new permission and exploit it, since no control over the signature will be applied.

A.12 Rule 16 - Erase data in WebView cache

Google Guideline. *If your application accesses sensitive data with a* `WebView`*, you may want to use the* `clearCache()` *method to delete any files stored locally. You can also use server-side headers such as* `no-cache` *to indicate that an application should not cache particular content.*

Pseudo-code. The Rule 16 pseudo-code is shown in Algorithm 16.

Rule 16: Erase data in webview cache

```
begin
    webViews ← getAllWebViewVars()
    foreach webView in webViews do
        if not usesClearCache(webView) then
            Rule 16 is not respected.
        end
    end
end
```

Attack. If an app using a `WebView` object does not clear its cache through the `clearCache()` method, any malicious code running within the app UID (e.g., third-party libraries) can access to the data saved in the cache.

A.13 Rule 17 - Avoid SQL injections

Google Guideline. *When accessing a content provider, use parameterized query methods such as* `query()`, `update()`, *and* `delete()` *to avoid potential SQL injection from untrusted sources. Note that using parameterized methods is not sufficient if the* `selection` *argument is built by concatenating user data prior to submitting it to the method.*
Don't have a false sense of security about the write permission. The write permission allows SQL statements that make it possible for some data to be confirmed using creative `WHERE` *clauses and parsing the results. For example, an attacker might probe for the presence of a specific phone number in a call log by modifying a row only if that phone number already exists. If the content provider data has predictable structure, the write permission may be equivalent to providing both reading and writing.*

Pseudo-code. The Rule 17 pseudo-code is shown in Algorithm 17.

Rule 17: Avoid SQL injections: use content providers

```
begin
    str ← "query"
    extendCP ← getClassesExtendCP()
    foreach obj in extendCP do
        methods ← getObjMethods(obj)
        foreach method in methods do
            if method = str then
                Rule 17 is not respected.
            end
        end
    end
end
```

Attack. If an app uses parameterized query methods to access one of its content providers, but the selection argument is built by concatenating user data, an attacker can launch SQL injection attacks.

A.14 Rule 18 - Prefer explicit intents

Google Guideline. *For activities and broadcast receivers, intents are the preferred mechanism for asynchronous IPC in Android. Depending on your application requirements, you might use* `sendBroadcast()`, `sendOrderedBroadcast()`, *or an explicit intent to a specific application component. For security purposes, explicit intents are preferred.*
Caution*: if you use an intent to bind to a* `Service`*, ensure that your app is secure by using an explicit intent. Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call* `bindService()` *with an implicit intent.*
Note that ordered broadcasts can be consumed by a recipient, so they may not be delivered to all applications. If you are sending an intent that must be delivered to a specific receiver, you must use an explicit intent that declares the receiver by name.
Senders of an intent can verify that the recipient has permission by specifying a non-null permission with the method call. Only applications with that permission receive the intent. If data within a broadcast intent may be sensitive, you should consider applying a permission to make sure that malicious applications can't register to receive those messages without appropriate permissions. In those circumstances, you may also consider invoking the receiver directly, rather than raising a broadcast.

Pseudo-code. The Rule 18 pseudo-code is shown in Algorithm 18.

Rule 18: Prefer explicit intents

```
begin
    bindNames ← getBindNamesIntents()
    startService ← getStartServiceIntents()
    sendOrdBcast ← getSendOrdBcastIntents()
    startActivity ← getStartActivityIntents()
    intents ← bindNames + startService +
     sendOrdBcast + startActivity
    foreach intent in intents do
        if not isExplicit(intent) then
            Rule 18 is not respected.
        end
    end
end
```

Attack. The attack is the same as for Rule 1.

A.15 Rule 19 - Use IP networking

Google guideline. *Some applications use localhost network ports for handling sensitive IPC. You should not use this approach because these interfaces are accessible by other applications on the device. Instead, use an Android IPC mechanism where authentication is possible, such as with a* `Service`*. Binding to* `INADDR_ANY` *is worse than using loopback because then your application may receive requests from anywhere.*

Pseudo-code. The rule we formalized is shown in Algorithm 19.

Rule 19: Use IP networking

```
begin
    arr ← ["INADDR_ANY", "localhost",
     "127.0.0.1"]
    javaCode ← getAllJavaCode()
    foreach word in javaCode do
        if word in arr then
            Rule 19 is not respected.
        end
    end
end
```

Attack. A malicious app can connect to the same localhost network ports as legitimate apps and intercept the messages they exchange.

A.16 Rule 20 - Use services

Google guideline. *A* `Service` *is often used to supply functionality for other applications to use. Each service class must have a corresponding* `<service>` *declaration in its manifest file.*
By default, services are not exported and cannot be invoked by any other application. However, if you add any intent filters to

the service declaration, it is exported by default. It's best if you explicitly declare the `android:exported` *attribute to be sure it behaves as you'd like. Services can also be protected using the* `android:permission` *attribute. By doing so, other applications need to declare a corresponding* `<uses-permission>` *element in their own manifest to be able to start, stop, or bind to the service.*
A service can protect individual IPC calls into it with permissions, by calling `checkCallingPermission()` *before executing the implementation of that call. You should use the declarative permissions in the manifest, since those are less prone to oversight.* ***Caution****: don't confuse client and server permissions; ensure that the called app has appropriate permissions and verify that you grant the same permissions to the calling app.*

Pseudo-code. The rule we formalized is shown in Algorithm 20.

Rule 20: Use services

```
begin
    services ← getAllServices()
    foreach service in services do
        if hasIntentFilter(service)
        or isExported(service) then
            if hasPermission(service) then
                if not checksCallingPerm(service)
                 then
                    Rule 20 is not respected.
                end
            end
        end
    end
end
```

Attack. If a `Service` is *exported*, a malicious app can interact with it by sending malicious `Intent` messages, that compromise the `Service` runtime execution.

A.17 Rule 21 - Use telephony networking

Google guideline. *The SMS protocol was primarily designed for user-to-user communication and is not well-suited for apps that want to transfer data. Due to the limitations of SMS, you should use Google Cloud Messaging (GCM) and IP networking for sending data messages from a web server to your app on a user device.*
Beware that SMS is neither encrypted nor strongly authenticated on either the network or the device. In particular, any SMS receiver should expect that a malicious user may have sent the SMS to your application. Don't rely on unauthenticated SMS data to perform sensitive commands. Also, you should be aware that SMS may be subject to spoofing and/or interception on the network. On the Android-powered device itself, SMS messages are transmitted as broadcast intents, so they may be read or captured by other applications that have the `READ_SMS` *permission.*

Pseudo-code. The rule we formalized is shown in Algorithm 21.

Attack. A malicious app declaring the SMS related permissions can intercept and modify messages targeting a legitimate app or it can even send a malicious SMS to the SMS `Receiver` of a legitimate app.

Rule 21: Use telephony networking

```
begin
    arr ← ["SEND_SMS", "READ_SMS",
     "RECEIVE_SMS"]
    permissions ← getAllPermissions()
    foreach perm in permissions do
        if perm in arr then
            Rule 21 is not respected.
    end
end
```

A.18 Rule 22 - Use cryptography

Google guideline.

Use a secure random number generator, `SecureRandom`*, to initialize any cryptographic keys generated by* `KeyGenerator`*. Use of a key that is not generated with a secure random number generator significantly weakens the strength of the algorithm and may allow offline attacks.*
If you need to store a key for repeated use, use a mechanism, such as `KeyStore`*, that provides a mechanism for long term storage and retrieval of cryptographic keys.*

Pseudo-code. The rule we formalized is shown in Algorithm 22.

Rule 22: Use cryptography

```
begin
    keyGens ← getAllKeyGenVars()
    secRands ← getAllSecRandVars()
    foreach keyGen in keyGens do
        if not initsWithAny(keyGen, secRands)
         then
            Rule 22 is not respected.
        end
    end
end
```

Attack. When keys are not generated through secure random number generators, a malicious app can infer the value of such keys and decrypt any sensitive data previously encrypted by the legitimate app.

A.19 Rule 23 - Use broadcast receivers

Google Guideline. *A* `BroadcastReceiver` *handles asynchronous requests initiated by an* `Intent`*.*
By default, receivers are exported and can be invoked by any other application. If your `BroadcastReceiver` *is intended for use by other applications, you may want to apply security permissions to receivers using the* `<receiver>` *element within the application manifest. This prevents applications without appropriate permissions from sending an intent to the* `BroadcastReceiver`*.*

Pseudo-code. The Rule 23 pseudo-code is shown in Algorithm 23.

Attack. Any malicious app can create an intent which can trigger an exported receiver not protected by a permission.

For instance, let's consider an exported and not protected receiver which sends an SMS to a phone number received as an extra parameter of the triggering intent. A malicious

Rule 23: Use broadcast receivers

```
begin
    receivers ← getAllBcastReceivers()
    foreach receiver in receivers do
        if isExported(receiver) then
            if not hasPermission(receiver) then
                Rule 23 is not respected.
            end
        end
    end
end
```

application could trigger the receiver by sending intents with a premium rate SMS number. Thus, it would force users to send messages without their consent, stealing them money.

A.20 Rule 24 - Dynamically load code

Google guideline. *We strongly discourage loading code from outside of your application APK. Doing so significantly increases the likelihood of application compromise due to code injection or code tampering. It also adds complexity around version management and application testing. It can also make it impossible to verify the behavior of an application, so it may be prohibited in some environments.*
If your application does dynamically load code, the most important thing to keep in mind about dynamically-loaded code is that it runs with the same security permissions as the application APK. The user makes a decision to install your application based on your identity, and the user expects that you provide any code run within the application, including code that is dynamically loaded. The major security risk associated with dynamically loading code is that the code needs to come from a verifiable source. If the modules are included directly within your APK, they cannot be modified by other applications. This is true whether the code is a native library or a class being loaded using `DexClassLoader`*. Many applications attempt to load code from insecure locations, such as downloaded from the network over unencrypted protocols or from world-writable locations such as external storage. These locations could allow someone on the network to modify the content in transit or another application on a user's device to modify the content on the device.*

Pseudo-code. The rule we formalized is shown in Algorithm 24.

Rule 24: Dynamically load code

```
begin
    str ← "DexClassLoader"
    javaCode ← getAllJavaCode()
    foreach word in javaCode do
        if word = str then
            Rule 24 is not respected.
        end
    end
end
```

Attack. A malicious app can launch a code injection attack through which it modifies the code that a legitimate app will dynamically load. This aim can be achieved if the code is saved in the external storage, is downloaded from a remote location (and, thus, intercepted and modified).

A.21 Rule 25 - Common problems with hostname verification

Google guideline. ***Caution****: Replacing* `HostnameVerifier` *can be very dangerous if the other virtual host is not under your control, because a man-in-the-middle attack could direct traffic to another server without your knowledge.*
If you are still sure you want to override hostname verification, here is an example that replaces the verifier for a single `URLConnection` *with one that still verifies that the hostname is at least on expected by the app:*

```
// Create an HostnameVerifier that hardwires the
// expected hostname. Note that is different than
// the URL's hostname: example.com versus
// example.org
HostnameVerifier verifier = new HostnameVerifier(){
  @Override
  public boolean verify(String hostname, SSLSession
    session) {
    HostnameVerifier hv=HttpsURLConnection.
   getDefaultHostnameVerifier();
    return hv.verify("example.com", session);
  }
};

// Tell the URLConnection to use our
// HostnameVerifier
URL url = new URL("https://example.org/");
HttpsURLConnection urlConnection =
  (HttpsURLConnection)url.openConnection();
urlConnection.setHostnameVerifier(verifier);
InputStream in = urlConnection.getInputStream();
copyInputStreamToOutputStream(in, System.out);
```

Listing 9. problems with hostname verification

Pseudo-code. The rule we formalized is shown in Algorithm 25.

Rule 25: Common problems with hostname verification

```
begin
    connections ← getAllHttpsUrlConnections()
    foreach connection in connections do
        if hasSetHostnameVerifier(connection)
          then
            Rule 25 is not respected.
        end
    end
end
```

Attack. A malicious app can perform a man-in-the-middle attack by redirecting the traffic, originally sent to a legitimate server, towards another malicious one.

A.22 Rule 26 - Warnings about using SSLSocket directly

Google guideline. ***Caution****:* `SSLSocket` *does not perform hostname verification. It is up to your app to do its own hostname verification, preferably by calling* `getDefaultHostnameVerifier()` *with the expected hostname. Further beware that* `HostnameVerifier.verify()`

doesn't throw an exception on error but instead returns a boolean result that you must explicitly check.
Here is an example showing how you can do this. It shows that when connecting to gmail.com port 443 without SNI support, you'll receive a certificate for mail.google.com. This is expected in this case, so check to make sure that the certificate is indeed for mail.google.com:

```
// Open SSLSocket directly to gmail.com
SocketFactory sf = SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) sf.createSocket("
   gmail.com", 443);
HostnameVerifier hv = HttpsURLConnection.
   getDefaultHostnameVerifier();
SSLSession s = socket.getSession();

// Verify that the certicate hostname is for
// mail.google.com. This is due to lack of SNI
// support in the current SSLSocket.
if (!hv.verify("mail.google.com", s)) {
  throw new SSLHandshakeException("Expected mail.
   google.com, found " + s.getPeerPrincipal());
}

// At this point SSLSocket performed certificate
// verification and we have performed hostname
// verification, so it is safe to proceed.

// ... use socket ...
socket.close();
```

Listing 10. Warnings about using SSLSocket directly

Pseudo-code. The rule we formalized is shown in Algorithm 26.

Rule 26: Warnings about using SSLSocket directly

```
begin
    sslSessions ← getAllSslSessions()
    verifiers ← getAllHostnameVerifiers()
    foreach ver in verifiers do
        if not verifiesWithAny(ver, sslSessions)
         then
            Rule 26 is not respected.
        end
    end
end
```

Attack. A malicious app can launch a man-in-the-middle attack against an app that does not use HTTPS or SSL at all. Moreover, if the victim app does not verify the certificate sent by a server, the attacker can even pretend to the remote server and establish a communication with the victim app.

A.23 Rule 27 - Configure CAs for debugging

Google guideline. *When debugging an app that connects over HTTPS, you may want to connect to a local development server, which does not have the SSL certificate for your production server. In order to support this without any modification to your app's code, you can specify debug-only CAs, which are trusted only when* `android:debuggable` *is true, by using* `debug-overrides`*. Normally, IDEs and build tools set this flag automatically for non-release builds.*
This is safer than the usual conditional code because, as a security precaution, app stores do not accept apps which are marked debuggable.
`res/xml/network_security_config.xml`*:*

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <debug-overrides>
    <trust-anchors>
      <certificates src="@raw/debug_cas"/>
    </trust-anchors>
  </debug-overrides>
</network-security-config>
```

Listing 11. Configure CAs for debugging

Pseudo-code. The rule we formalized is shown in Algorithm 27.

Rule 27: Configure CAs for debugging

```
begin
    str ← "networkSecurityConfig"
    element1 ← "<network-security-config>"
    element2 ← "<debug-overrides>"
    app ← getManifestApplicationElement()
    appAttrs ← getAttrs(app);
    respected ← False
    foreach attr in appAttrs do
        if attr = str then
            confElements ← getNetSecElements()
            if element1 in confElements then
                if element2 in confElements then
                    respected ← True
                    break
                end
            end
        end
    end
    if not respected then
        Rule 27 is not respected.
    end
end
```

Attack. Using conditional code to handle connection to a local development server could lead to mistakes in production builds. If developers forget this conditional code, or this conditional code is not well managed, then an attacker could exploit these mistakes and perform a man-in-the-middle attack.

A.24 Rule 28 - Opt out of cleartext traffic

Google guideline. ***Note****: the guidance in this section applies only to apps that target Android 8.1 (API level 27) or lower. Starting with Android 9 (API level 28), cleartext support is disabled by default.*
Applications intending to connect to destinations using only secure connections can opt-out of supporting cleartext (using the unencrypted HTTP protocol instead of HTTPS) to those destinations. This option helps prevent accidental regressions in apps due to changes in URLs provided by external sources such as backend servers. See `NetworkSecurityPolicy.isCleartextTrafficPermitted()` *for more details.*
For example, an app may want to ensure that all connections to `secure.example.com` *are always done over HTTPS to protect sensitive traffic from hostile networks.*
`res/xml/network_security_config.xml`*:*

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="false">
    <domain includeSubdomains="true">
     secure.example.com
    </domain>
  </domain-config>
</network-security-config>
```

Listing 12. Opt out of cleartext traffic

Pseudo-code. The rule we formalized is shown in Algorithm 28.

Rule 28: Opt out of cleartext traffic

```
begin
    s1 ← "networkSecurityConfig"
    s2 ← "cleartextTrafficPermitted"
    element ← "<domain-config>"
    app ← getManifestApplicationElement()
    appAttrs ← getAttrs(app);
    foreach attr in appAttrs do
        if attr = s1 then
            confElements ← getNetSecElements()
            if element in confElements then
                dcAttrs ← getAttrs(element)
                foreach dcAttr in dcAttrs do
                    if dcAttr.name = s2 then
                        if dcAttr.value then
                            Rule 28 is not respected.
                        end
                    end
                end
            end
        end
    end
end
```

Attack. With the `cleartextTrafficPermitted` flag set to `true`, any connection using HTTP is allowed. Thus, an attacker can eavesdrop the cleartext content of any communication established by the victim app.

A.25 Rule 30 - Deprecated cryptographic functionality

Google guideline. *The following subsections describe deprecated functionality that you should no longer use in your app.*

A.25.1 Bouncy Castle algorithms

A number of algorithms from the "Bouncy Castle provider"[2] that are also provided by another provider have been depracated in Android P. This only affects cases where the implementation from the Bouncy Castle provider is explicitly requested, such as `Cipher.getInstance("AES/CBC/PKCS7PADDING", "BC"` or `Cipher.getInstance("AES/CBC/PKCS7PADDING", Security.getProvider("BC"))`. Requesting a specific provider is discouraged, so if you follow that guideline this deprecation should not affect you.

2. https://www.bouncycastle.org/

A.25.2 Password-based encryption ciphers without an IV

Password-based encryption (PBE) ciphers that require an initialization vector (IV) can obtain it from the key, if it's suitably constructed, or from an explicitly-passed IV. When passing a PBE key that doesn't contain an IV and no explicit IV, the PBE ciphers on Android currently assume an IV of zero.
When using PBE ciphers, always pass an explicit IV, as shown in the following code snippet:

```
SecretKey key = ...;
Cipher cipher = Cipher.getInstance(
 "PBEWITHSHA256AND256BITAES-CBC-BC");
byte[] iv = new byte[16];
new SecureRandom().nextBytes(iv);
cipher.init(Cipher.ENCRYPT_MODE, key, new
    IvParameterSpec(iv));
```

Listing 13. Password-based encryption ciphers without an IV

Pseudo-code. The rule we formalized is shown in Algorithm 30.

Rule 30: Deprecated cryptographic functionality

```
begin
    ciphers ← getAllCipherGetInstance()
    foreach cipher in ciphers do
        if hasSecondArgument(cipher) then
            Rule 30 is not respected.
        end
        if hasPBE(cipher) then
            if not hasInit(cipher) then
                Rule 30 is not respected.
            end
        end
    end
end
```

Attack. When deprecated and insecure crytographic algorithms are used, a malicious app can decrypt any sensitive data previously encrypted by the legitimate app.

A.26 Rule 31 - Migrate existing data

Google guideline. *If a user updates their device to use Direct Boot mode, you might have existing data that needs to get migrated to device encrypted storage. Use `Context.moveSharedPreferencesFrom()` and `Context.moveDatabaseFrom()` to migrate preference and database data between credential encrypted storage and device encrypted storage.*
Use your best judgment when deciding what data to migrate from credential encrypted storage to device encrypted storage. You should not migrate private user information, such as passwords or authorization tokens, to device encrypted storage. In some scenarios, you might need to manage separate sets of data in the two encrypted stores.

Pseudo-code. The rule we formalized is shown in Algorithm 31.

Attack. A malicious app, that has access to the device encrypted storage, could scan the device encrypted storage searching for private information, such as passwords or authorization tokens.

Rule 31: Migrate existing data

```
begin
    arr ← ["moveSharedPreferencesFrom",
     "moveDatabaseFrom"]
    methods ← getAllCalledMethods()
    foreach method in methods do
        if method in arr then
            Rule 31 is not respected.
        end
    end
end
```

A.27 Rule 32 - Access device encrypted storage

Google guideline. *Use device encrypted storage only for information that must be accessible during Direct Boot mode. Do not use device encrypted storage as a general-purpose encrypted store.*

Pseudo-code. The rule we formalized is shown in Algorithm 32.

Rule 32: Access device encrypted storage

```
begin
    methods ← getAllCalledMethods()
    str ← "createDeviceProtectedStorageContext"
    foreach method in methods do
        if method = str then
            Rule 32 is not respected.
        end
    end
end
```

Attack. A malicious app, that has access to the device encrypted storage, could scan the device encrypted storage searching for private information, such as passwords or authorization tokens.

APPENDIX B
RULES VIOLATIONS EXAMPLES FROM REAL-WORLD APP

B.1 Rule 1 - Show an app chooser

```
public boolean onOptionsItemSelected(MenuItem
    menuItem) {
...
} else {
    com.hinkhoj.dictionary.b.a.a(getActivity(), "
    Share", "Scrabble Game", "");
    Intent intent = new Intent();
    intent.setAction("android.intent.action.SEND");
    intent.putExtra("android.intent.extra.SUBJECT",
    "Hinkhoj's Scrabble Game");
    intent.putExtra("android.intent.extra.TEXT", "
    Hey I love to play the Scrabble Game of Hinkhoj\
    nPlease download the app from here: http://dict.
    hinkhoj.com/install-app.php\n\n");
    intent.setType("text/plain");
    startActivity(intent);
    return true;
    }
}
```

B.2 Rule 2 - Content provider access control

```
<provider android:name="com.hinkhoj.dictionary.
    WordSearch.wordsearch.view.
    WordDictionaryProvider" android:authorities="com
    .hinkhoj.dictionary.wordsearch.provider.words"/>
```

B.3 Rule 3 - Provide the right permissions

```
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
    <uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
    <uses-permission android:name="android.
    permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.
    permission.CAMERA"/>
    <uses-permission android:name="android.
    permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.
    permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.
    permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.
    permission.USE_CREDENTIALS"/>
    <uses-permission android:name="android.
    permission.INTERNET"/>
    <uses-permission android:name="android.
    permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="com.android.
    vending.BILLING"/>
    <uses-permission android:name="android.
    permission.AUTHENTICATE_ACCOUNTS"/>
    <uses-permission android:name="com.google.
    android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="android.
    permission.WAKE_LOCK"/>
    <uses-permission android:name="android.
    permission.VIBRATE"/>
    <uses-permission android:name="android.
    permission.GET_TASKS"/>
```

```
    <uses-permission android:name="android.
    permission.READ_CONTACTS"/>
    <uses-permission android:name="android.
    permission.TYPE_APPLICATION_OVERLAY"/>
    <uses-permission android:name="android.
    permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.
    permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.
    permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-feature android:name="android.hardware.
    touchscreen" android:required="false"/>
    <uses-feature android:name="android.hardware.
    portrait" android:required="false"/>
    <uses-feature android:name="android.hardware.
    camera" android:required="false"/>
    <uses-permission android:name="android.
    permission.RECEIVE_SMS" android:protectionLevel=
    "signature"/>
```

B.4 Rule 4 - Use intents to defer permissions

```
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
<uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
...
<uses-permission android:name="android.permission.
    ACCESS_COARSE_LOCATION"
<uses-permission android:name="android.permission.
    CAMERA"
...
<uses-permission android:name="android.permission.
    GET_ACCOUNTS"
...
<uses-permission android:name="android.permission.
    READ_CONTACTS"
...
<uses-permission android:name="android.permission.
    READ_EXTERNAL_STORAGE"
...
<uses-permission android:name="android.permission.
    RECEIVE_SMS" android:protectionLevel="signature"
```

B.5 Rule 5 - Use SSL traffic

```
public static String a(String str) throws
    IOException {
        InputStream inputStream;
        HttpURLConnection httpURLConnection = (
    HttpURLConnection) new URL(str).openConnection()
    ;
        httpURLConnection.setRequestProperty("Accept
    -Encoding", "gzip, deflate");
        httpURLConnection.setConnectTimeout(
    CBConstant.HTTP_TIMEOUT);
        httpURLConnection.setReadTimeout(CBConstant.
    HTTP_TIMEOUT);
        httpURLConnection.connect();
        ...
```

B.6 Rule 6 - Use HTML message channels

```
private void n() {
        this.s.getSettings().setJavaScriptEnabled(
    true);
        this.s.addJavascriptInterface(this, "PayU");
        this.s.getSettings().
    setSupportMultipleWindows(true);
        ...
```

B.7 Rule 7 - Use WebView objects carefully

```
private static String c(Context context, boolean z)
    {
        ...
        synchronized (d.class) {
            if (f1618a != null) {
                return f1618a;
            }
            ...
            try {
                try {
                    f1618a = a(context, "android.
    webkit.WebSettings", "android.webkit.WebView");
                } catch (Exception unused2) {
                    f1618a = a(context, "android.
    webkit.WebSettingsClassic", "android.webkit.
    WebViewClassic");
                }
            } catch (Exception unused3) {
                WebView webView = new WebView(
    context.getApplicationContext());
                f1618a = webView.getSettings().
    getUserAgentString();
                webView.destroy();
            }
            return f1618a;
        }
    }
```

B.8 Rule 8 - Store private data within internal storage

```
public String readFileInputStream(Context context,
    String str, int i) {
        String str2 = "";
        try {
            if (!new File(context.getFilesDir(), str
    ).exists()) {
                context.openFileOutput(str, i);
            }
            FileInputStream openFileInput = context.
    openFileInput(str);
            while (true) {
                int read = openFileInput.read();
                if (read == -1) {
                    break;
                }
                str2 = str2 + Character.toString((
    char) read);
            }
            openFileInput.close();
            ...
```

B.9 Rule 9 - Share data securely across apps

```
public void onClick(View view) {
            Intent intent = new Intent("org.
    openintents.action.PICK_FILE");
            intent.setData(Uri.parse("file://" + ((
    Object) WordListActivity.this.b.getText())));
            intent.putExtra("org.openintents.extra.
    BUTTON_TEXT", WordListActivity.this.getString(R.
    string.EXPORT));
            if (WordListActivity.this.b()) {
                WordListActivity.this.
    startActivityForResult(intent, 2);
            } else {
                WordListActivity.this.a();
            }
        }
```

B.10 Rule 10 - Use scoped directory access

```
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
    <uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
    ...
    <uses-permission android:name="android.
    permission.READ_EXTERNAL_STORAGE"
    <uses-permission android:name="android.
    permission.WRITE_EXTERNAL_STORAGE"
    ...
```

B.11 Rule 11 - Store only non-sensitive data in cache files

```
private static a b(Context context, String str)
    throws IOException, XmlPullParserException {
        b bVar = new b(str);
        XmlResourceParser loadXmlMetaData = context.
    getPackageManager().resolveContentProvider(str,
    128).loadXmlMetaData(context.getPackageManager()
    , "android.support.FILE_PROVIDER_PATHS");
        if (loadXmlMetaData == null) {
            throw new IllegalArgumentException("
    Missing android.support.FILE_PROVIDER_PATHS meta
    -data");
        }
        while (true) {
            int next = loadXmlMetaData.next();
            if (next == 1) {
                return bVar;
            }
            if (next == 2) {
                String name = loadXmlMetaData.
    getName();
                File file = null;
                String attributeValue =
    loadXmlMetaData.getAttributeValue(null, "name");
                String attributeValue2 =
    loadXmlMetaData.getAttributeValue(null, "path");
                if ("root-path".equals(name)) {
                    file = b;
                } else if ("files-path".equals(name)
    ) {
                    file = context.getFilesDir();
                } else if ("cache-path".equals(name)
    ) {
                    file = context.getCacheDir();
                } else if ("external-path".equals(
    name)) {
                    file = Environment.
    getExternalStorageDirectory();
                }
                ...
```

B.12 Rule 12 - Use SharedPreferences in private mode

```
private static SharedPreferences a(Context context)
    {
        return context.getSharedPreferences("
    multidex.version", Build.VERSION.SDK_INT < 11 ?
    0 : 4);
    }
```

B.13 Rule 14 - Check validity of data

```
private static a b(Context context, String str)
    throws IOException, XmlPullParserException {
        b bVar = new b(str);
        XmlResourceParser loadXmlMetaData = context.
    getPackageManager().resolveContentProvider(str,
    128).loadXmlMetaData(context.getPackageManager()
    , "android.support.FILE_PROVIDER_PATHS");
        if (loadXmlMetaData == null) {
            throw new IllegalArgumentException("
    Missing android.support.FILE_PROVIDER_PATHS meta
    -data");
        }
        while (true) {
            int next = loadXmlMetaData.next();
            if (next == 1) {
                return bVar;
            }
            if (next == 2) {
                String name = loadXmlMetaData.
    getName();
                File file = null;
                String attributeValue =
    loadXmlMetaData.getAttributeValue(null, "name");
                String attributeValue2 =
    loadXmlMetaData.getAttributeValue(null, "path");
                if ("root-path".equals(name)) {
                    file = b;
                } else if ("files-path".equals(name)
    ) {
                    file = context.getFilesDir();
                } else if ("cache-path".equals(name)
    ) {
                    file = context.getCacheDir();
                } else if ("external-path".equals(
    name)) {
                    file = Environment.
    getExternalStorageDirectory();
                } else if ("external-files-path".
    equals(name)) {
                    File[] a2 = b.a(context, (String
    ) null);
                    if (a2.length > 0) {
                        file = a2[0];
                    }
                    ...
```

B.14 Rule 16 - Erase data in WebView cache

```
public void fillOTPOnBankPage() {
        try {
            if (this.i != null && !TextUtils.isEmpty
    (this.ah) && this.i.has(getString(d.g.
    cb_fill_otp))) {
                WebView webView = this.s;
                webView.loadUrl("javascript:" + this
    .i.getString(getString(d.g.cb_fill_otp)) + "(\""
     + this.ah + "\",\"url\")");
                this.ah = null;
            }
        } catch (JSONException e) {
            com.google.a.a.a.a.a.a.a(e);
        }
    }
```

B.15 Rule 17 - Avoid SQL injections

```
public Cursor query(Uri uri, String[] strArr, String
     str, String[] strArr2, String str2) {...}
```

B.16 Rule 18 - Prefer explicit intents

```
public static boolean a(Context context, String str,
     d dVar) {
        Intent intent = new Intent("android.support.
    customtabs.action.CustomTabsService");
        if (!TextUtils.isEmpty(str)) {
            intent.setPackage(str);
        }
        return context.bindService(intent, dVar, 33)
    ;
    }
```

B.17 Rule 19 - Use IP networking

```
private f(c cVar) {
        this.f1570a = new Object();
        this.b = Executors.newFixedThreadPool(8);
        this.c = new ConcurrentHashMap();
        this.g = (c) j.a(cVar);
        try {
            this.d = new ServerSocket(0, 8,
    InetAddress.getByName("127.0.0.1"));
            this.e = this.d.getLocalPort();
            ...
```

B.18 Rule 20 - Use services

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
<uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
...
<service android:name="com.firebase.jobdispatcher.
    GooglePlayReceiver" android:permission="com.
    google.android.gms.permission.
    BIND_NETWORK_TASK_SERVICE" android:exported="
    true"
...
```

B.19 Rule 21 - Use telephony networking

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
<uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
...
<uses-permission android:name="android.permission.
    RECEIVE_SMS" android:protectionLevel="signature"
...
```

B.20 Rule 22 - Use cryptography

```
private static void generateKey(KeyGenParameterSpec
    keyGenParameterSpec) throws
    GeneralSecurityException {
        try {
            KeyGenerator instance = KeyGenerator.
    getInstance("AES", ANDROID_KEYSTORE);
            instance.init(keyGenParameterSpec);
            instance.generateKey();
        } catch (ProviderException e) {
            throw new GeneralSecurityException(e.
    getMessage(), e);
        }
    }
```

B.21 Rule 23 - Use broadcast receivers

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/
    apk/res/android" android:versionCode="93"
    android:versionName="8.3.3.7" android:
    installLocation="auto" package="HinKhoj.
    Dictionary">
<uses-sdk android:minSdkVersion="16" android:
    targetSdkVersion="26"/>
...
<receiver android:name="com.hinkhoj.dictionary.
    receiver.OfflineAnalyticsReceiver" android:
    enabled="true" android:exported="true"
...
```

B.22 Rule 24 - Dynamically load code

```
public static ahi a(Context context, String str,
    String str2, boolean z) {
...
ahi.d = new DexClassLoader(file.getAbsolutePath(),
    cacheDir.getAbsolutePath(), null, ahi.f2925a.
    getClassLoader());
a(file);
ahi.a(cacheDir, "1505450608132");
a(String.format("%s/%s.dex", cacheDir, "
    1505450608132"));
if (((Boolean) bnr.f().a(bqz.br)).booleanValue() &&
    !ahi.s) {
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction("android.intent.action.
    USER_PRESENT");
    intentFilter.addAction("android.intent.action.
    SCREEN_OFF");
    ahi.f2925a.registerReceiver(new a(ahi, null),
    intentFilter);
    ahi.s = true;
...
```

B.23 Rule 25 - Common problems with hostname verification

```
public final class zzd extends HttpsURLConnection {
    private final zze zzhf;
    private final HttpsURLConnection zzhg;

    /* access modifiers changed from: package-
    private */
    public zzd(HttpsURLConnection httpsURLConnection
    , zzbg zzbg, zzat zzat) {
        super(httpsURLConnection.getURL());
        this.zzhg = httpsURLConnection;
        this.zzhf = new zze(httpsURLConnection, zzbg
    , zzat);
    }
    ...
```

B.24 Rule 26 - Warnings about using SSLSocket directly

```
private final Socket i() {
    ...
    try {
        SSLSocket sSLSocket = (SSLSocket)
    SSLCertificateSocketFactory.getDefault(60000,
    sSLSessionCache).createSocket(host, port);
        if (HttpsURLConnection.
    getDefaultHostnameVerifier().verify(host,
    sSLSocket.getSession())) {
            return sSLSocket;
        }
```

```
        String valueOf4 = String.valueOf(this.f);
        StringBuilder sb2 = new StringBuilder(39 +
    String.valueOf(valueOf4).length());
        sb2.append("Error while verifying secure
    socket to ");
        sb2.append(valueOf4);
        throw new bbl(sb2.toString());
    } catch (UnknownHostException e4) {
        String valueOf5 = String.valueOf(host);
        throw new bbl(valueOf5.length() != 0 ? "
    unknown host: ".concat(valueOf5) : new String("
    unknown host: "), e4);
        ...
```

B.25 Rule 28 - Opt out of cleartext traffic

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    ...
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">localhost
        </domain>
    </domain-config>
</network-security-config>
```

B.26 Rule 29 - Choose a recommended algorithm

```
public final class amy implements amx<Cipher> {
    /* Return type fixed from 'java.lang.Object' to
    match base method */
    @Override // com.google.android.gms.internal.amx
    public final /* synthetic */ Cipher a(String str
    , Provider provider) throws
    GeneralSecurityException {
        return provider == null ? Cipher.getInstance
    (str) : Cipher.getInstance(str, provider);
    }
```

B.27 Rule 30 - Deprecated cryptographic functionality

```
private static Cipher getCipher() throws
    NoSuchPaddingException, NoSuchAlgorithmException
     {
        if (Util.SDK_INT == 18) {
            try {
                return Cipher.getInstance("AES/CBC/
    PKCS5PADDING", "BC");
            } catch (Throwable unused) {
            }
        }
        return Cipher.getInstance("AES/CBC/
    PKCS5PADDING");
    }
```

B.28 Rule 32 - Access device encrypted storage

```
public static Context
    createDeviceProtectedStorageContext(@NonNull
    Context context) {
        if (Build.VERSION.SDK_INT >= 24) {
            return context.
    createDeviceProtectedStorageContext();
        }
        return null;
    }
```

APPENDIX C
THE 100 APKS ANALYZED IN THE EXPERIMENTS

ID	Name	Package Name	Size	Downloads
1	8 Ball Pool	com.miniclip.eightballpool	13.6 MB	500,000,000+
2	Abs Workout A6W	fitness.flatstomach.homeworkout.absworkout	4.98 MB	10,000,000+
3	Amazon Kindle	com.amazon.kindle	33.54 MB	100,000,000+
4	Amazon Prime Video	com.amazon.avod.thirdpartyclient	28.99 MB	100,000,000+
5	Amazon Shopping	com.amazon.mShop.android.shopping	48.93 MB	100,000,000+
6	Android Accessibility Suite	com.google.android.marvin.talkback	3.96 MB	5,000,000,000+
7	AppLock	com.domobile.applock	7.53 MB	50,000,000+
8	Ball Mayhem!	com.clement.ballmayhem	23.11 MB	10,000,000+
9	BitLife	com.candywriter.bitlife	14.37 MB	10,000,000+
10	Bitmoji	com.bitstrips.imoji	17.05 MB	100,000,000+
11	BookMyShow	com.bt.bms	22.52 MB	50,000,000+
12	Candy Crush Friends Saga	com.king.candycrush4	7.19 MB	10,000,000+
13	Candy Crush Saga	com.king.candycrushsaga	7.35 MB	500,000,000+
14	Cloud Print	com.google.android.apps.cloudprint	3.12 MB	1,000,000,000+
15	Coin Master	com.moonactive.coinmaster	10.59 MB	50,000,000+
16	Color Bump 3D	com.colorup.game	22.43 MB	100,000,000+
17	DH Texas Poker	com.droidhen.game.poker	7.44 MB	10,000,000+
18	Discord	com.discord	13.54 MB	100,000,000+
19	Doodle Toy!™	com.doodletoy	1.83 MB	10,000,000+
20	DoorDash	com.dd.doordash	15.07 MB	10,000,000+
21	Dream League Soccer	com.firsttouchgames.dlsa	7.61 MB	10,000,000+
22	Drive and Park	com.parking.game	12.91 MB	50,000,000+
23	Drum Pad Machine	com.agminstruments.drumpadmachine	20.07 MB	50,000,000+
24	Endomondo	com.endomondo.android	13.03 MB	10,000,000+
25	English Hindi Dictionary	HinKhoj.Dictionary	13.86 MB	10,000,000+
26	FindNow	com.ratelekom.findnow	27.27 MB	10,000,000+
27	Fire Balls 3D	com.NikSanTech.FireDots3D	23.1 MB	50,000,000+
28	Flick Shoot	net.mobilecraft.football	4.62 MB	10,000,000+
29	GO Weather	com.gau.go.launcherex.gowidget.weatherwidget	13.79 MB	50,000,000+
30	Google Play Games	com.google.android.play.games	7.87 MB	1,000,000,000+
31	Google Sheets	com.google.android.apps.docs.editors.sheets	21.1 MB	500,000,000+
32	GooglePlayServices for AR	com.google.ar.core	5.29 MB	100,000,000+
33	Granny	com.dvloper.granny	6.88 MB	100,000,000+
34	Grass Cut	com.fullfat.bw	31.73 MB	10,000,000+
35	Grubhub	com.grubhub.android	20.15 MB	10,000,000+
36	HOOKED	tv.telepathic.hooked	19.14 MB	10,000,000+
37	Happy Color™	com.pixel.art.coloring.color.number	25.91 MB	50,000,000+
38	Happy Glass	com.game5mobile.lineandwater	21.73 MB	100,000,000+
39	Helix Jump	com.h8games.helixjump	30.08 MB	100,000,000+
40	Hole.io	io.voodoo.holeio	22.3 MB	50,000,000+
41	Homescapes	com.playrix.homescapes	9.81 MB	100,000,000+
42	Hotspot Shield Free	hotspotshield.android.vpn	17.7 MB	100,000,000+
43	Hulu	com.hulu.plus	14.24 MB	50,000,000+
44	Ice Age Village	com.gameloft.android.ANMP.GloftIAHM	6.31 MB	50,000,000+
45	Idle Supermarket Tycoon	com.codigames.market.idle.tycoon	15.3 MB	10,000,000+
46	Instagram	com.instagram.android	26.24 MB	1,000,000,000+
47	Life360	com.life360.android.safetymapd	22.25 MB	50,000,000+
48	Light-It Up	com.crazylabs.light.it.up	21.76 MB	10,000,000+
49	Logo Quiz	logos.quiz.companies.game	13.79 MB	50,000,000+
50	Lucky Day	com.luckyday.app	35.4 MB	10,000,000+
51	Lyft	me.lyft.android	66.48 MB	10,000,000+
52	Magic Tiles 3	com.youmusic.magictiles	13.08 MB	100,000,000+
53	Microsoft Outlook	com.microsoft.office.outlook	42.89 MB	100,000,000+
54	Netflix	com.netflix.mediaclient	14.3 MB	500,000,000+
55	News Break	com.particlenews.newsbreak	12.56 MB	10,000,000+
56	OfferUp	com.offerup	28.49 MB	10,000,000+

57	Oppa doll	com.percent.mybest	19.71 MB	10,000,000+
58	Paint Pop 3D	com.magjg.roundhit	23.15 MB	50,000,000+
59	Paper.io 2	io.voodoo.paper2	28.04 MB	100,000,000+
60	PayPal	com.paypal.android.p2pmobile	34.0 MB	100,000,000+
61	Pinterest	com.pinterest	2.97 MB	100,000,000+
62	Pluto TV	tv.pluto.android	15.29 MB	10,000,000+
63	Polysphere	com.playgendary.polyspherecoolgame	16.09 MB	50,000,000+
64	Postmates	com.postmates.android	7.38 MB	5,000,000+
65	Roblox	com.roblox.client	4.84 MB	100,000,000+
66	Samsung Notes	com.samsung.android.app.notes	18.77 MB	500,000,000+
67	Sea Battle	com.byril.seabattle	10.99 MB	10,000,000+
68	Skater Boy	com.game.SkaterBoy	10.07 MB	100,000,000+
69	Sky Map	com.google.android.stardroid	459.99 KB	50,000,000+
70	Skype	com.skype.android.access	846.58 KB	1,000,000,000+
71	Smart Lock	ukzzang.android.app.protectorlite	7.19 MB	5,000,000+
72	Snowball.io	com.geishatokyo.snowballio	10.3 MB	10,000,000+
73	Solitaire	com.brainium.solitairefree	13.26 MB	10,000,000+
74	SoundCloud	com.soundcloud.android	22.02 MB	100,000,000+
75	Spotify	com.spotify.music	27.5 MB	500,000,000+
76	Stadium Horn	com.progimax.airhorn.free	3.55 MB	10,000,000+
77	Stickman Hook	com.mindy.grap1	11.38 MB	50,000,000+
78	Subway Surfers	com.kiloo.subwaysurf	12.0 MB	1,000,000,000+
79	Tank Stars	com.playgendary.tanks	14.17 MB	100,000,000+
80	TextNow	com.enflick.android.TextNow	43.72 MB	50,000,000+
81	The Weather Channel	com.weather.Weather	24.09 MB	100,000,000+
82	Tiles Hop	com.amanotes.beathopper	15.47 MB	100,000,000+
83	Tubi	com.tubitv	16.64 MB	50,000,000+
84	Twitter	com.twitter.android	1.85 MB	500,000,000+
85	Uber	com.ubercab	6.43 MB	500,000,000+
86	Video Collage Maker	com.scoompa.collagemaker.video	10.91 MB	10,000,000+
87	Waze	com.waze	17.74 MB	100,000,000+
88	Weeder Match	idle.weedermatch.casualgame	16.35 MB	1,000,000+
89	Wisepilot	org.microemu.android.se.appello.lp.Lightpilot	6.1 MB	5,000,000+
90	Wish	com.contextlogic.wish	23.2 MB	100,000,000+
91	Word Cookies!®	com.bitmango.go.wordcookies	14.95 MB	10,000,000+
92	Word Swipe	com.wordgame.puzzle.block.crush	13.29 MB	10,000,000+
93	Words Story	com.word.game.fun.puzzle.prison.escape.captain	8.41 MB	50,000,000+
94	Wordscapes	com.peoplefun.wordcross	25.8 MB	10,000,000+
95	YP	com.yellowpages.android.ypmobile	10.18 MB	50,000,000+
96	YouTube Kids	com.google.android.apps.youtube.kids	11.56 MB	100,000,000+
97	YouTube Music	com.google.android.apps.youtube.music	15.95 MB	100,000,000+
98	ZArchiver	ru.zdevs.zarchiver	501.69 KB	50,000,000+
99	ZEDGE™	net.zedge.android	39.13 MB	100,000,000+
100	Zombie Evil	com.feelingtouch.gnz	5.98 MB	10,000,000+

APPENDIX D
EVALUATION DATA

TABLE 2
Raw data of Fig. 4 together with average times (in seconds) for each rule.

Rule	Apps	Violations	Avg. Time
1	87.0	477.0	407.200
2	21.0	30.0	0.061
3	3.0	36.0	3.058
4	72.0	225.0	0.059
5	96.0	1946.0	2.165
6	94.0	2390.0	2.261
7	84.0	2146.0	419.263
8	50.0	140.0	11.222
9	3.0	7.0	400.487
10	77.0	120.0	0.178
11	95.0	1818.0	10.766
12	74.0	160.0	9.805
13	43.0	43.0	19.909
14	32.0	108.0	7.241
15	0.0	0.0	0.075
16	91.0	2459.0	330.776
17	89.0	392.0	29.707
18	87.0	907.0	364.488
19	34.0	51.0	0.677
20	59.0	380.0	38.930
21	3.0	5.0	0.072
22	22.0	42.0	253.346
23	88.0	426.0	2.767
24	52.0	87.0	0.522
25	37.0	52.0	278.396
26	80.0	472.0	236.193
27	0.0	0.0	0.085
28	0.0	0.0	0.082
29	96.0	2821.0	2.413
30	35.0	70.0	280.774
31	0.0	0.0	9.236
32	64.0	146.0	9.200

TABLE 3
Raw data for Fig. 5 (Rules 1-16).

AppID	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
1	13.499	0.017	0.019	0.016	0.270	0.210	11.614	1.864	13.772	0.019	1.914	1.855	5.285	7.755	0.017	11.809
2	386.269	0.074	0.088	0.068	1.765	5.663	590.866	22.222	374.935	0.233	22.959	22.119	2.296	0.502	0.096	580.138
3	456.160	0.077	0.084	0.328	1.538	2.209	701.892	13.349	458.985	0.086	12.025	11.690	17.213	13.730	0.085	678.697
4	812.221	0.381	0.276	0.052	5.753	10.052	596.821	48.741	440.674	0.130	9.378	9.134	10.989	13.637	0.059	345.384
5	991.242	0.139	0.148	0.111	2.061	0.956	1001.714	9.493	968.438	0.128	8.912	8.258	29.333	0.584	0.144	1001.077
6	280.801	0.089	0.071	0.049	0.930	0.362	287.250	5.099	296.378	0.147	5.150	4.925	0.074	0.000	0.063	77.082
7	868.317	0.199	0.192	0.171	2.431	1.756	650.757	11.763	702.769	0.230	12.639	12.082	24.532	16.207	0.193	629.568
8	143.798	0.050	0.088	0.042	1.430	1.384	321.765	6.975	101.380	0.047	6.280	6.063	17.209	0.440	0.045	337.622
9	129.707	0.055	0.065	0.057	0.761	0.492	402.942	5.335	135.196	0.059	4.825	4.676	1.629	8.793	0.065	418.858
10	52.605	0.026	0.022	0.020	0.409	0.416	37.263	2.459	118.692	0.065	3.122	2.758	4.603	15.603	0.022	105.512
11	242.039	0.022	0.024	0.026	0.424	0.249	240.969	3.011	243.136	0.025	3.129	2.966	7.023	11.507	0.024	39.941
12	28.865	0.013	0.013	0.011	0.101	0.089	100.730	1.058	104.691	0.046	1.672	0.808	6.025	0.414	0.013	43.580
13	62.372	0.040	0.039	0.036	1.144	0.973	315.698	5.570	144.068	0.154	5.766	5.486	5.130	0.421	0.047	78.338
14	428.836	0.068	0.073	0.064	1.811	5.400	543.648	19.939	513.811	0.160	20.448	19.754	29.454	0.718	0.082	516.787
15	328.724	0.057	4.740	0.062	1.250	1.909	325.779	8.364	328.869	0.062	8.831	8.475	12.473	0.461	0.059	326.284
16	559.938	0.070	0.090	0.065	1.881	4.739	521.670	18.028	516.029	0.172	18.623	18.017	26.195	0.461	0.081	512.803
17	92.847	0.100	0.065	0.103	0.840	0.605	91.075	6.331	92.159	0.066	6.435	6.292	10.669	5.358	0.065	92.148
18	369.880	0.055	0.067	0.054	1.373	2.829	326.708	10.249	364.045	0.155	12.199	11.654	15.755	21.080	0.059	372.628
19	438.522	0.090	0.093	0.117	0.996	0.626	441.985	7.047	445.287	0.095	7.257	7.079	25.780	5.042	0.093	239.298
20	188.602	0.068	0.086	0.070	0.664	0.468	168.859	7.388	336.397	0.181	8.683	8.453	22.781	5.852	0.083	188.382
21	266.774	0.022	0.055	0.020	0.194	0.158	56.957	1.714	249.136	0.027	1.581	1.442	4.690	0.391	0.017	59.183
22	15.533	0.009	0.010	0.013	0.095	0.100	50.523	1.065	57.915	0.019	1.072	0.942	3.267	0.382	0.009	58.235
23	87.170	0.027	0.028	0.025	0.808	0.538	287.438	2.982	87.999	0.030	3.191	2.984	5.237	0.441	0.028	87.484
24	45.718	0.026	0.059	0.024	0.543	1.151	67.835	4.491	68.364	0.106	5.149	4.200	3.695	0.381	0.025	23.217
25	435.885	0.047	0.315	0.038	3.351	1.459	424.720	18.872	438.389	0.085	4.733	4.592	9.635	0.461	0.050	73.460
26	283.371	0.098	0.105	0.108	2.429	3.209	257.791	20.700	270.126	0.110	21.263	20.649	34.407	20.029	0.107	260.107
27	229.400	0.018	0.028	0.017	0.523	0.390	248.193	1.646	283.277	0.021	1.720	1.593	5.844	0.457	0.019	82.566
28	57.245	0.018	0.031	0.022	0.481	0.609	335.107	3.120	124.473	0.072	3.239	3.050	8.758	25.393	0.019	272.186
29	410.349	0.076	0.086	0.071	2.176	5.770	599.448	19.807	413.295	0.234	23.549	22.515	2.231	0.531	0.102	603.568
30	1086.222	0.007	0.007	0.006	0.209	0.230	890.145	0.718	1104.692	0.008	0.744	0.712	0.008	12.500	0.007	889.082
31	343.184	0.061	0.060	0.053	1.503	2.232	343.371	9.393	345.171	0.061	9.631	9.355	14.186	0.672	0.059	339.374
32	274.670	0.027	0.028	0.030	0.911	1.075	239.482	3.154	259.306	0.089	3.499	3.265	5.638	11.800	0.040	70.399
33	264.621	0.044	0.039	0.038	0.346	0.346	265.571	5.897	284.231	0.093	6.037	5.878	13.029	8.269	0.038	67.228
34	271.265	0.048	0.044	0.045	1.023	0.968	269.717	4.970	270.388	0.046	5.107	4.937	14.937	0.442	0.044	68.526
35	20.772	0.016	0.024	0.018	0.092	0.064	19.573	1.107	20.153	0.021	1.119	1.055	2.723	1.050	0.015	21.011
36	159.944	0.094	0.098	0.079	0.773	0.845	152.219	4.375	171.266	0.302	5.250	4.986	32.362	0.624	0.104	140.315
37	67.967	0.060	0.062	0.063	0.818	0.810	64.228	6.488	68.469	0.064	6.707	6.543	26.997	0.702	0.062	64.828
38	64.054	0.089	0.093	0.087	1.140	0.991	62.638	7.228	63.301	0.095	7.486	7.263	34.449	105.377	0.094	61.830
39	15.883	0.016	0.019	0.010	0.067	0.085	15.240	0.550	14.510	0.029	0.556	0.572	0.948	0.428	0.015	3.808
40	139.545	0.063	0.087	0.062	0.904	0.720	118.381	6.891	112.477	0.067	6.177	5.963	20.935	0.431	0.065	50.879
41	28.441	0.010	0.037	0.008	0.130	0.175	29.385	0.448	34.044	0.041	0.401	0.164	0.016	0.403	0.016	31.207
42	7.161	0.022	0.544	0.025	0.388	0.161	6.749	1.873	7.068	0.023	1.913	1.845	10.504	0.580	0.024	6.592
43	569.478	0.091	0.104	0.092	1.670	0.915	567.311	7.927	570.807	0.105	8.071	7.909	17.215	8.248	0.103	366.391
44	394.415	0.080	0.085	0.069	1.784	5.792	571.373	22.931	391.106	0.200	23.781	23.074	2.378	0.455	0.101	575.440
45	309.806	0.052	0.055	0.051	1.016	0.822	247.830	8.944	263.247	0.142	8.761	8.521	28.134	0.430	0.062	183.506
46	216.784	0.152	0.151	0.147	1.663	1.663	204.199	16.975	205.028	0.149	17.476	16.957	56.537	3.506	0.147	203.098
47	997.697	0.042	0.566	0.042	11.989	32.514	983.153	63.118	963.961	0.347	109.787	111.612	32.468	0.652	0.087	701.590
48	308.950	0.053	0.104	0.065	0.546	0.702	368.547	31.404	423.567	0.578	33.893	33.317	5.507	0.690	0.059	427.921
49	242.396	0.071	0.072	0.069	0.468	0.597	239.740	5.966	241.775	0.073	6.081	5.938	4.540	0.892	0.070	239.432
50	781.344	0.088	0.118	0.090	1.581	0.996	725.389	13.387	728.182	0.277	13.654	13.163	21.986	8.395	0.099	502.626
51	747.754	0.088	0.101	0.088	2.065	3.313	753.364	17.442	733.482	0.206	18.819	17.578	28.280	16.256	0.100	743.667
52	1885.873	0.059	0.622	0.055	16.802	31.923	1819.659	73.327	1750.750	0.671	54.385	49.497	174.701	0.491	0.910	1812.757
53	805.923	0.117	0.154	0.117	2.007	2.271	631.578	18.173	818.462	0.316	18.629	18.096	51.148	9.905	0.139	591.393
54	114.859	0.072	0.053	0.047	1.084	1.317	392.094	7.381	135.518	0.050	6.706	6.395	7.203	0.448	0.048	381.535
55	109.742	0.051	0.056	0.048	1.004	2.097	109.978	8.994	111.806	0.057	9.308	8.958	12.729	0.456	0.055	107.139
56	87.893	0.041	0.036	0.035	0.845	1.012	46.698	4.369	127.319	0.117	5.184	4.960	13.307	3.728	0.036	82.328
57	349.557	0.054	0.058	0.053	0.697	0.529	339.846	8.060	350.433	0.060	8.226	8.041	24.527	0.523	0.058	186.977
58	1131.209	0.105	0.138	0.114	1.836	1.814	1076.332	13.037	1115.358	0.328	14.302	13.728	34.667	7.901	0.123	879.092
59	89.470	0.042	0.044	0.043	1.187	1.304	285.367	6.914	88.472	0.048	7.222	6.917	7.036	7.332	0.046	283.814
60	318.725	0.046	0.057	0.046	1.142	1.335	314.998	7.134	339.753	0.126	8.468	8.121	25.196	24.800	0.057	411.012
61	3294.419	0.094	1.245	0.093	72.810	11.373	3510.744	116.046	3591.011	5.694	73.104	25.993	237.564	10.483	0.220	3390.502
62	353.146	0.067	0.073	0.065	1.860	3.259	528.916	12.670	353.599	0.073	12.274	13.216	25.694	0.493	0.085	301.801
63	330.582	0.051	0.058	0.046	1.010	1.318	366.896	6.661	252.323	0.126	6.891	6.528	7.204	8.281	0.057	317.566
64	845.586	0.093	0.108	0.093	0.835	0.782	845.520	12.280	850.864	0.256	12.493	12.139	33.988	0.480	0.124	653.862
65	356.494	0.065	0.064	0.057	1.245	4.071	363.368	12.715	361.222	0.066	13.086	12.666	18.424	0.462	0.064	357.821
66	369.300	0.054	0.069	0.055	1.774	2.041	450.926	8.445	366.559	0.064	7.675	7.337	10.381	14.683	0.062	411.230
67	302.860	0.045	0.056	0.040	1.460	1.437	298.862	7.123	297.169	0.107	7.436	7.106	9.174	0.446	0.056	279.208
68	468.569	0.064	0.158	0.073	2.226	5.325	356.499	19.505	344.769	0.073	8.778	8.425	22.366	7.259	0.170	416.350
69	361.751	0.056	0.064	0.054	0.788	0.585	359.973	6.140	360.831	0.064	6.320	6.146	10.675	0.409	0.063	359.721
70	7.278	0.009	0.010	0.014	0.083	0.145	7.240	0.599	7.296	0.011	0.618	0.601	2.128	0.407	0.009	7.139
71	562.793	0.076	0.088	0.079	1.532	2.255	589.064	11.041	580.023	0.207	12.025	11.064	21.007	10.925	0.087	351.857
72	235.042	0.042	0.045	0.043	0.443	0.390	33.892	4.690	235.205	0.045	4.810	4.662	12.285	0.417	0.044	33.768
73	289.618	0.061	290.827	0.062	0.900	0.449	385.205	13.569	307.237	0.071	14.163	13.508	18.447	4.865	0.070	193.807
74	123.117	0.022	0.054	0.027	0.346	0.296	281.620	2.601	90.771	0.026	2.248	2.154	6.840	9.482	0.025	244.109
75	27.004	0.006	0.022	0.009	0.034	0.045	27.046	1.043	27.704	0.020	0.460	0.389	0.005	0.379	0.006	29.045
76	441.621	0.072	0.084	0.072	1.302	1.165	472.319	7.163	470.029	0.192	8.502	8.169	43.623	30.840	0.091	191.612
77	2908.375	0.134	0.136	0.125	1.535	1.734	2498.390	13.015	2202.974	0.123	11.645	11.360	42.263	83.185	0.123	0.000
78	611.094	0.080	0.106	0.078	1.475	1.101	632.239	9.278	659.801	0.235	10.885	10.558	27.773	10.172	0.093	430.950
79	9.380	0.015	0.012	0.015	0.164	0.078	226.381	1.232	34.425	0.026	1.218	1.147	3.979	1.872	0.012	32.417
80	709.857	0.071	0.088	0.072	0.946	0.890	677.662	4.728	701.649	0.202	5.659	5.426	37.274	39.828	0.088	464.068
81	448.916	0.056	0.129	0.061	0.515	1.137	236.480	7.773	450.601	0.118	8.562	7.803	19.669	4.134	0.060	241.889
82	738.194	0.097	0.125	0.094	2.159	1.484	508.957	11.709	722.577	0.255	13.864	13.404	44.669	0.501	0.124	531.904
83	301.004	0.036	0.080	0.031	0.685	0.431	298.546	5.186	295.786	0.142	5.421	5.135	4.134	0.487	0.038	284.695
84	78.703	0.050	0.054	0.050	1.042	0.829	276.485	6.196	78.701	0.056	6.451	6.224	6.605	7.495	0.054	74.866
85	234.139	0.016	0.016	0.022	0.220	0.110	257.420	1.846	285.953	0.046	1.960	1.762	4.253	12.634	0.024	84.545
86	204.800	0.055	0.063	0.048	1.281	1.884	139.333	7.714	278.285	0.125	9.166	8.796	11.276	0.631	0.063	197.496
87	2.706	0.013	0.014	0.018	0.012	0.012	2.723	0.063	2.727	0.015	0.064	0.063	0.012	0.458	0.013	2.691
88	109.066	0.062	0.071	0.055	1.039	0.695	176.341	6.630	90.734	0.064	7.669	7.526	10.303	7.208	0.079	163.336
89	815.502	0.060	0.430	0.049	10.038	10.127	843.106	48.023	864.522	0.586	50.403	33.428	15.120	0.486	0.132	860.237
90	521.450	0.067	0.083	0.069	2.013	5.868	516.482	19.382	505.525	0.225	19.910	19.260	28.955	0.491	0.081	502.191
91	553.818	0.079	0.091	0.073	1.773	5.406	566.057	20.779	566.265	0.082	19.672	20.660	32.242	0.506	0.088	534.920
92	214.936	0.027	0.027	0.025	0.587	0.693	334.068	3.133	133.776	0.067	3.723	3.547	10.607	0.450	0.027	376.738
93	1374.110	0.187	0.216	0.199	1.791	1.736	1483.836	7.102	1484.659	0.215	7.326	7.119	72.174	20.804	0.214	1264.427
94	13.764	0.015	0.016	0.018	0.169	0.161	12.754	2.232	38.965	0.059	2.754	2.090	3.056	0.391	0.016	38.425
95	32.204	0.028	0.031	0.030	0.275	0.201	29.021	3.710	32.328	0.032	3.788	3.711	10.361	2.101	0.031	29.239
96	372.929	0.032	0.048	0.030	0.480	0.293	319.589	2.329	303.434	0.040	2.399	2.223	8.016	0.453	0.043	83.541
97	26.814	0.012	0.012	0.015	0.023	0.023	5.146	0.780	7.172	0.013	0.801	0.782	0.009	0.556	0.011	5.158
98	425.733	0.063	0.077	0.062	1.610	0.975	397.575	8.545	348.686	0.069	7.713	7.520	25.457	0.443	0.071	401.655
99	450.578	0.058	0.072	0.061	1.677	0.643	384.194	8.765	366.513	0.065	8.999	8.709	14.496	5.457	0.073	238.249
100	280.478	0.021	0.022	0.026	0.260	0.218	63.285	2.502	289.278	0.072	2.525	2.332	1.926	7.667	0.029	83.810

TABLE 4
Raw data for Fig. 5 (Rules 17-32).

AppID	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	R32
1	2.180	12.984	0.090	1.757	0.018	11.774	0.344	0.096	11.808	11.805	0.014	0.018	0.348	11.652	1.902	1.890
2	34.911	388.714	1.141	30.381	0.087	369.608	7.406	0.747	356.335	365.088	0.073	0.088	3.143	382.089	19.994	19.747
3	2.021	448.184	0.776	3.150	0.083	470.984	9.252	0.701	440.923	478.935	0.324	0.092	3.206	432.705	12.153	13.449
4	101.860	238.742	0.683	0.811	0.058	130.551	5.954	0.398	127.415	161.065	0.202	0.050	1.803	104.979	9.840	9.320
5	1.146	976.121	0.802	6.572	0.127	967.458	13.044	0.786	1013.646	1005.447	0.142	0.127	2.941	927.554	8.543	8.453
6	0.952	379.776	0.277	0.860	0.058	135.538	4.167	0.235	504.107	106.201	0.269	0.050	0.988	311.284	5.184	5.031
7	1.825	683.491	1.438	11.569	0.220	655.040	3.406	1.124	634.241	630.841	0.377	0.192	4.850	652.591	12.508	12.355
8	0.727	160.683	0.417	1.015	0.046	71.706	3.959	0.340	131.297	121.579	0.224	0.046	1.630	82.458	7.269	6.994
9	0.587	129.518	0.400	2.277	0.056	209.320	0.987	0.328	116.498	416.014	0.169	0.065	1.635	139.311	4.765	4.724
10	15.407	36.579	0.166	0.022	0.023	51.515	0.295	0.439	104.405	99.771	0.061	0.023	0.663	238.897	3.069	2.781
11	13.416	242.239	0.119	0.027	0.023	39.838	0.289	0.118	40.171	39.720	0.052	0.023	0.516	40.021	3.059	3.014
12	7.555	25.363	0.084	0.013	0.013	74.543	0.212	0.263	102.936	79.698	0.040	0.013	0.294	25.936	0.722	0.716
13	120.174	108.703	0.602	0.661	0.048	128.428	2.575	0.328	61.262	111.036	0.089	0.061	1.757	126.865	4.922	4.860
14	1.381	334.419	1.101	25.071	0.096	314.417	5.171	0.542	316.195	321.893	0.159	0.072	2.860	332.947	17.976	17.804
15	55.970	130.146	0.605	28.539	0.059	124.930	6.400	0.499	125.379	123.427	0.136	0.060	2.168	126.956	8.601	8.519
16	63.151	517.134	0.999	28.711	0.080	338.456	8.181	0.697	307.434	309.245	0.160	0.082	3.261	312.916	18.746	18.478
17	143.102	93.644	0.517	30.752	0.064	92.637	1.098	0.368	89.814	90.262	0.185	0.064	1.651	90.423	6.418	6.374
18	130.634	326.178	0.604	0.045	0.056	178.343	7.262	1.437	158.248	190.686	0.161	0.074	2.491	141.596	10.527	11.159
19	19.818	443.747	0.660	8.739	0.092	235.561	2.531	0.530	239.333	239.115	0.145	0.092	2.165	238.767	7.267	7.188
20	25.874	225.737	0.508	7.330	0.076	152.763	0.977	0.529	199.059	191.491	0.083	0.090	1.805	143.364	8.053	8.540
21	15.369	232.517	0.116	0.023	0.017	60.741	0.695	0.173	34.579	15.460	0.061	0.018	0.436	217.875	1.540	1.439
22	11.557	57.894	0.049	0.015	0.009	13.725	0.288	0.056	14.001	14.040	0.068	0.009	0.193	15.727	0.889	0.850
23	35.132	88.587	0.243	0.032	0.029	84.779	0.280	0.176	288.393	87.517	0.058	0.029	0.895	286.062	3.047	3.006
24	25.550	22.528	0.183	0.024	0.026	35.380	0.319	0.740	63.314	67.566	0.096	0.083	1.151	60.440	4.012	3.994
25	4.152	315.393	0.269	0.046	0.046	165.923	0.376	0.222	98.955	68.992	0.057	0.088	0.988	160.152	4.801	4.659
26	37.334	255.338	1.148	36.742	0.111	253.096	1.630	0.848	279.282	276.403	0.137	0.115	4.583	274.311	24.226	22.641
27	9.832	25.655	0.158	0.016	0.019	23.881	0.264	0.115	251.777	24.774	0.060	0.019	0.562	24.347	1.420	1.399
28	8.202	76.870	0.143	0.025	0.018	43.307	0.662	0.128	364.194	116.557	0.161	0.039	0.912	73.238	2.766	2.738
29	95.861	408.463	1.110	27.122	0.086	386.868	4.785	0.828	396.950	398.976	0.128	0.087	3.440	397.566	23.318	22.958
30	13.321	897.522	0.053	0.017	0.007	894.651	0.117	0.054	906.752	874.848	0.039	0.007	0.332	1084.470	0.734	0.729
31	110.682	340.511	0.535	0.049	0.060	137.458	9.776	0.443	137.489	136.117	0.097	0.060	1.875	138.829	9.598	9.495
32	17.886	238.158	0.311	0.038	0.028	55.794	0.844	0.197	243.879	56.404	0.077	0.028	1.011	35.634	2.854	2.825
33	7.904	353.228	0.242	3.992	0.046	155.504	4.121	0.326	74.599	92.573	0.080	0.038	1.104	155.756	5.214	5.146
34	82.929	272.524	0.395	24.017	0.044	67.144	2.659	0.340	68.283	68.442	0.071	0.044	1.562	68.595	5.066	5.036
35	1.897	20.202	0.106	0.069	0.041	19.830	0.928	0.056	14.766	4.218	0.045	0.016	0.149	4.256	0.944	0.933
36	81.296	184.756	0.672	0.073	0.089	143.571	1.031	0.833	164.654	154.870	0.101	0.105	2.701	148.280	4.529	4.476
37	395.461	66.775	0.689	138.230	0.062	63.965	0.418	0.694	64.935	62.777	0.077	0.061	2.708	64.969	6.700	6.608
38	459.056	63.149	1.034	229.421	0.094	62.211	1.155	0.999	62.114	61.933	0.130	0.093	4.079	62.244	7.475	7.350
39	0.257	3.205	0.022	0.241	0.011	3.069	0.181	0.022	3.027	3.041	0.030	0.013	0.103	3.011	0.490	0.482
40	116.068	131.204	0.810	117.452	0.074	72.993	0.529	0.607	52.478	105.918	0.115	0.129	2.835	117.466	6.144	6.042
41	0.228	29.004	0.057	0.004	0.017	23.107	0.109	0.011	5.254	5.049	0.057	0.011	0.078	4.959	0.062	0.060
42	24.890	6.919	0.196	23.558	0.022	6.606	0.564	0.177	6.972	12.779	0.076	0.055	0.782	22.261	2.260	2.133
43	12.219	568.957	0.849	16.302	0.104	362.766	1.531	0.747	366.716	369.340	0.119	0.103	3.140	370.929	8.020	7.957
44	28.008	385.919	1.149	28.974	0.083	373.135	10.624	0.760	355.239	368.570	0.120	0.093	3.737	364.100	20.653	20.400
45	15.374	238.499	0.664	18.585	0.062	229.930	3.689	0.504	246.971	391.033	0.058	0.079	2.283	459.800	8.795	8.522
46	201.016	205.623	1.444	277.141	0.149	200.961	1.570	1.365	205.341	208.271	0.173	0.148	5.353	202.676	17.411	17.152
47	23.392	400.157	2.947	19.855	0.205	489.141	5.370	2.642	363.558	347.283	0.062	0.455	10.608	453.058	63.743	13.560
48	14.592	215.409	2.085	10.856	0.367	172.674	0.504	0.289	245.856	106.290	0.066	0.275	6.116	227.184	6.776	7.036
49	12.669	241.775	0.318	14.732	0.072	39.338	0.950	0.302	39.257	38.755	0.101	0.073	1.221	39.208	6.083	6.028
50	17.624	746.767	0.920	30.663	0.116	512.226	1.187	0.697	535.229	510.943	0.118	0.253	3.604	707.153	13.575	11.723
51	27.247	541.929	1.346	23.122	0.101	548.057	6.276	0.929	741.279	530.530	0.121	0.100	3.807	594.311	14.620	14.446
52	28.052	1779.568	3.517	26.992	0.126	1535.115	2.272	1.208	1570.636	1594.791	0.069	0.690	26.989	1569.826	28.621	87.213
53	32.872	806.241	1.271	31.958	0.163	613.104	14.738	0.836	598.273	805.643	0.122	0.155	4.179	849.101	18.810	18.320
54	20.062	152.496	0.445	13.382	0.048	144.556	5.799	0.490	152.209	128.623	0.046	0.056	2.142	162.771	6.578	6.455
55	28.917	110.893	0.594	0.054	0.056	105.229	2.932	0.490	106.613	103.223	0.054	0.055	2.007	104.946	9.126	9.004
56	8.036	48.223	0.357	0.035	0.036	111.400	0.472	0.410	113.635	44.604	0.041	0.036	1.098	89.731	5.527	5.080
57	38.012	359.094	0.439	37.863	0.058	186.449	5.496	0.442	179.222	187.315	0.054	0.059	1.888	179.187	8.251	8.149
58	12.273	1048.325	1.170	27.500	0.138	907.864	5.851	1.067	896.248	922.830	0.105	0.139	4.125	1121.230	14.205	13.872
59	12.779	87.988	0.379	15.230	0.045	80.883	0.337	0.369	83.061	80.887	0.042	0.044	1.784	82.546	7.041	6.996
60	43.892	0.000	0.000	45.019	0.000	0.000	0.502	0.000	0.000	0.000	0.046	0.000	0.000	0.000	0.000	0.000
61	4.404	2336.564	1.437	22.454	0.113	434.026	1.878	0.628	420.880	626.138	0.094	0.125	3.086	445.890	11.297	10.985
62	1.828	334.801	1.108	36.462	0.080	324.811	5.616	0.706	507.400	300.489	0.060	0.088	3.571	469.740	14.008	13.704
63	16.292	102.135	0.579	18.519	0.050	120.388	5.004	0.503	137.631	82.823	0.044	0.061	2.208	161.035	6.823	6.616
64	1.027	838.216	0.716	11.510	0.109	661.336	4.291	0.615	645.776	663.349	0.087	0.175	2.777	698.156	12.312	10.796
65	27.733	358.204	0.861	28.224	0.063	154.510	2.779	0.513	153.617	155.682	0.062	0.064	2.407	153.736	13.064	12.992
66	0.793	172.906	0.679	29.482	0.061	216.997	5.288	0.684	150.468	215.062	0.057	0.061	2.385	204.001	8.721	8.472
67	0.626	78.551	0.476	16.266	0.048	126.995	5.713	0.481	112.524	78.349	0.048	0.056	2.207	155.239	7.404	7.224
68	0.640	340.043	0.677	14.721	0.071	265.954	4.835	0.411	302.990	320.639	0.062	0.073	1.881	379.530	48.506	49.776
69	0.835	362.011	0.692	15.025	0.064	158.875	0.625	0.463	160.729	158.573	0.055	0.063	2.083	160.270	6.289	6.238
70	0.444	22.483	0.169	0.087	0.026	33.204	0.153	0.150	30.541	30.427	0.008	0.039	0.459	34.186	0.885	0.688
71	1.313	551.629	1.018	27.371	0.096	351.853	5.744	0.998	358.332	364.481	0.073	0.088	3.274	373.374	11.250	11.149
72	0.359	235.879	0.280	11.806	0.044	33.762	0.601	0.264	33.675	34.025	0.048	0.045	0.971	33.674	4.793	4.761
73	0.647	594.114	0.400	6.795	0.060	160.196	0.886	0.791	206.119	227.803	0.058	0.069	1.818	172.914	13.973	13.640
74	3.214	50.408	0.159	0.188	0.024	45.529	0.340	0.530	119.806	106.631	0.019	0.024	0.681	41.192	2.194	2.159
75	0.170	209.727	0.059	0.069	0.010	26.969	0.138	0.055	27.088	26.467	0.005	0.027	0.107	211.563	0.374	0.432
76	1.303	413.070	0.966	72.375	0.092	263.097	0.786	0.724	217.240	207.190	0.065	0.091	3.579	209.679	7.397	7.295
77	2.082	3706.256	1.376	336.460	0.139	2343.648	1.736	1.062	2931.539	0.000	0.117	0.137	4.917	2295.661	11.739	13.157
78	1.172	635.939	0.935	103.919	0.102	426.606	1.359	0.795	426.282	414.435	0.071	0.196	3.296	426.264	9.520	9.438
79	3.552	18.375	0.172	10.060	0.012	7.289	0.298	0.061	7.335	7.360	0.010	0.012	0.347	7.266	1.031	1.023
80	1.121	707.881	0.983	264.504	0.087	460.104	1.180	0.734	474.562	463.779	0.074	0.079	2.792	465.762	5.783	5.496
81	0.431	534.348	0.608	14.255	0.100	244.390	0.568	0.314	246.757	247.309	0.050	0.115	2.220	248.926	8.542	7.643
82	1.190	710.845	1.037	125.550	0.141	520.835	2.275	0.814	513.451	519.179	0.102	0.106	3.173	535.825	11.950	11.785
83	0.378	281.962	0.420	47.660	0.046	89.160	3.698	0.264	44.182	93.792	0.030	0.084	1.303	117.004	5.492	5.195
84	0.653	79.416	0.541	62.119	0.054	74.952	4.080	0.380	75.930	74.481	0.050	0.053	1.901	74.750	6.384	6.315
85	1.213	31.744	0.066	4.324	0.016	49.398	1.523	0.102	306.636	98.399	0.013	0.037	0.330	305.675	1.895	1.778
86	0.694	151.802	0.695	128.815	0.072	167.415	2.697	0.430	163.814	144.360	0.047	0.055	1.892	182.287	9.120	8.898
87	0.012	2.753	0.013	3.277	0.013	2.701	0.730	0.012	2.717	2.710	0.011	0.013	0.029	2.712	0.065	0.063
88	0.743	95.673	0.811	52.766	0.083	167.884	1.172	0.451	88.410	158.663	0.054	0.069	2.192	109.549	6.862	6.784
89	1.157	840.143	2.560	85.135	0.289	626.580	4.174	0.677	616.003	621.050	0.052	0.087	5.145	679.123	51.264	50.667
90	0.955	335.688	1.246	106.864	0.081	301.895	3.964	0.628	290.505	297.510	0.063	0.081	3.442	309.399	20.029	19.635
91	1.456	347.131	1.201	248.162	0.090	336.737	4.944	0.733	332.097	326.624	0.068	0.090	3.547	340.962	18.720	18.488
92	5.798	144.687	0.225	2.175	0.027	192.014	0.475	0.242	124.882	200.903	0.021	0.026	0.812	160.391	3.752	3.570
93	2.557	1464.997	1.805	297.738	0.219	1273.933	6.956	1.591	1265.955	1303.709	0.173	0.214	6.046	1301.402	7.289	7.213
94	2.141	45.370	0.243	2.749	0.045	40.513	0.498	0.360	39.613	22.802	0.015	0.016	0.480	12.352	1.874	1.853
95	0.373	33.349	0.208	37.430	0.030	29.505	1.037	0.203	29.913	29.768	0.023	0.031	0.909	29.494	3.796	3.764
96	2.888	255.628	0.206	4.212	0.036	49.357	0.739	0.151	280.305	53.724	0.020	0.027	0.770	147.795	2.393	2.255
97	0.160	8.343	0.024	0.822	0.011	5.179	0.138	0.023	5.274	5.173	0.009	0.011	0.082	5.254	0.793	0.781
98	0.750	366.397	0.693	99.348	0.077	150.168	0.692	0.639	219.976	185.898	0.058	0.091	2.372	155.417	7.650	7.837
99	0.779	435.182	0.675	111.852	0.080	176.735	0.788	0.453	243.895	185.753	0.055	0.095	2.552	171.618	8.442	8.787
100	2.229	283.736	0.275	3.701	0.023	79.288	0.914	0.427	48.213	22.271	0.019	0.021	0.549	226.768	2.456	2.352

TABLE 5
Raw data for the manual precision validation. For each app, we show the number of violations detected and the corresponding true positives for each rule.

	Instagram		Spotify		Wish		Idle Supermarket Tycoon		TextNow		Grass Cut		Samsung Notes		Twitter		Skype		Amazon Shopping	
Rules	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**	**Vio.**	**TPs**
R1	5	5	6	6	8	3	5	4	4	4	6	6	21	12	-	-	3	3	15	14
R2	-	-	3	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R3	-	-	-	-	-	-	6	1	-	-	-	-	27	19	-	-	-	-	-	-
R4	6	6	5	5	5	5	-	-	15	15	-	-	6	6	1	1	1	1	7	7
R5	18	13	17	15	32	20	36	36	16	16	10	10	24	22	2	1	-	-	73	64
R6	19	16	17	17	25	25	46	44	53	53	99	98	5	5	4	4	1	1	24	24
R7	22	20	41	38	57	56	36	36	71	71	94	94	13	13	3	3	2	2	22	22
R8	-	-	-	-	-	-	2	1	7	7	9	9	-	-	-	-	-	-	-	-
R9	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	-	-	-	-
R10	1	1	2	2	2	2	-	-	2	2	-	-	2	2	2	2	1	1	2	2
R11	60	60	12	12	14	14	31	31	27	27	55	55	30	30	-	-	-	-	48	48
R12	5	5	1	1	1	1	1	1	2	2	1	1	9	9	-	-	-	-	1	1
R13	1	1	1	1	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-
R14	-	-	1	0	-	-	-	-	2	2	-	-	17	13	-	-	-	-	7	2
R15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R16	22	22	44	44	59	58	40	40	77	77	101	101	44	44	3	3	2	2	28	28
R17	12	10	10	2	3	0	5	5	8	8	4	4	8	6	2	0	-	-	9	5
R18	3	3	8	4	14	11	8	6	-	-	-	-	24	8	-	-	-	-	13	13
R19	1	1	-	-	-	-	1	1	1	1	2	2	1	1	-	-	-	-	-	-
R20	8	8	2	2	-	-	-	-	5	4	-	-	1	1	2	2	-	-	-	-
R21	-	-	-	-	-	-	-	-	3	3	-	-	-	-	-	-	-	-	-	-
R22	1	1	1	1	2	1	1	0	3	2	1	0	2	2	-	-	-	-	5	1
R23	34	34	10	10	2	2	3	3	12	12	6	6	9	9	0	0	1	1	10	10
R24	-	-	-	-	2	2	-	-	-	-	1	1	-	-	-	-	-	-	3	3
R25	2	2	-	-	3	2	1	1	1	1	1	1	1	1	-	-	-	-	4	3
R26	2	2	4	2	7	4	4	0	7	5	14	13	8	8	4	4	-	-	12	12
R27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R28	1	1	-	-	-	-	-	-	1	1	-	-	-	-	-	-	1	1	-	-
R29	36	35	22	22	109	108	43	42	63	61	71	69	30	28	4	4	2	2	83	75
R30	-	-	1	1	8	8	1	1	11	11	1	1	-	-	-	-	-	-	11	11
R31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
R32	-	-	3	3	5	5	2	2	3	3	2	2	4	4	1	1	-	-	-	-