More Than Just a Random Number Generator! Unveiling the Security and Privacy Risks of Mobile OTP Authenticator Apps

Muhammad Ikram$^{1[0000-0003-2113-3390]\star}$, I Wayan Budi Sentana$^{1,2[0000-0003-3559-5123]}$, Hassan Asghar$^{1[0000-0001-6168-6497]}$, Mohamed Ali Kaafar$^{1[0000-0003-2714-0276]}$, and Michal Kepkowski$^{1[0000-0001-6364-4792]}$

[1] School of Computing, Macquarie University, NSW 2109, Australia
[2] Department of Information Technology, Politeknik Negeri Bali, Indonesia

Abstract. One-Time Passwords (OTPs) are a crucial component of multi-factor authentication (MFA) systems, providing additional security by requiring users to supply a dynamically generated code for authenticating to web services. The growth in smartphone usage has resulted in a shift from hardware tokens to mobile app-based OTP authenticators; however, these apps also present potential security and privacy threats.
In this paper, we present a comprehensive analysis of 182 publicly available OTP apps on Google Play. Our analysis entails an array of passive and active measurements meticulously designed to assess the security and privacy attributes inherent to each OTP application. We investigate the presence of suspicious libraries, usage of binary protections, access to root privileges, secure backup and cryptographic mechanisms, and protection against traffic interception, as well as gauge users' perceptions of the security and privacy features of OTP apps. Our experiments highlight several security and privacy weaknesses in instances of OTP apps. We observe that 28% of the analyzed apps are signed using a vulnerable version of the Android application signing mechanism. Over 40% of the OTP apps include third-party libraries leading to user information leakage to third-parties. 31.9% of the OTP applications are vulnerable to network interception, and only 13.2% possess the capability to detect devices that have been Jailbroken or rooted, which poses a significant concern. Our study highlights the need for better security and privacy guarantees in OTP apps and the importance of user awareness.

1 Introduction

OTPs are increasingly being used as part of the multi-factor authentication (MFA) paradigm, which provides an extra security layer to online systems rather than relying on passwords alone. In OTP-backed MFA systems, users are generally required to supply a dynamically generated four- or eight-digit one-time code in addition to their passwords. To ensure that the dynamic code reaches only the intended recipient, the OTP provider requires a secure transmission method. Initially, OTP delivery methods included traditional approaches like SMS or phone calls. However, for enhanced security, they have transitioned to dedicated hardware or software OTP authenticators. With increased smartphone usage, there has been a noticeable shift from hardware tokens to OTP authenticators based on mobile apps. Recent studies on MFA user experiences revealed that the most popular OTP code delivery method was SMS/email based, with 86% of the users reporting its use [10,7].

Despite the convenience, OTP apps also have potential threats to user security and privacy since the apps run on multipurpose smartphones. A report [48] shows that most mobile OTP apps have a design flaw that can be hacked: a threat actor can steal the OTP seed (the key used to generate OTP codes) through a privilege escalation attack via malware installed on the user's device, and then

[⋆] Corresponding author: muhammad.ikram@mq.edu.au produce identical OTP codes as the target's authenticator. Another report shows the increasing trend of OTP interception bots that target a bank's customers and have the ability to exfiltrate OTP codes to the attacker server [14]. Considering the vital role, the increasing trend, and the potential security and privacy threat of OTP apps, in this study, we conduct a comprehensive security and privacy analysis on mobile apps that generate OTP codes, which we call OTP apps. This study involves 182 apps publicly available in the Google Play Store and reveals several security and privacy flaws in OTP apps. Some of our main findings are summarized below. Other key takeaways from our analysis are highlighted throughout the paper.

- 51 (28.0%) OTP apps are signed using version 1 of Android's application signing mechanism which is known to be susceptible to Janus [31] and Master Key [27] vulnerabilities that allow attackers to inject a DEX file into an APK file without affecting the signatures and to insert malicious payloads into the package.
- 73 (40.1%) of OTP apps embed third-party libraries in their packages leading to user information leaking to the third party. We also found one OTP app embedding the Cross-Library Data Harvesting (XLDH) library which illegally extracts user information and sends it to third-party domains [60].
- 58 (31.9%) OTP apps are vulnerable to network interception enabling an attacker to intercept as well as tamper network traffic to and from the app.
- Only 24 (13.2%) OTP apps manage to detect Jailbroken or rooted devices as confirmed by our dynamic analysis, which allows an attacker to conduct reconnaissance to identify security vulnerabilities of OTP apps.
- 52 (28.6%) OTP apps failed to disclose their privacy policy to users and 12 (6.6%) of OTP apps failed to adhere to their privacy policy in terms of user information sharing with the third-parties.

Our findings indicate that many users of OTP apps may be unaware of the poor security and usability guarantees posed by these apps, despite promises made by the majority of them. To encourage further research in this area, we are making our dataset and analysis scripts available to the research community at `https://github.com/otpappsanalyzer/otpappsanalyzer`.

2 OTP Apps: Background and Discovery

We first provide an overview of OTP mechanisms and then present the methodology used for identifying Android apps on the Google Play that implement OTP authentication.

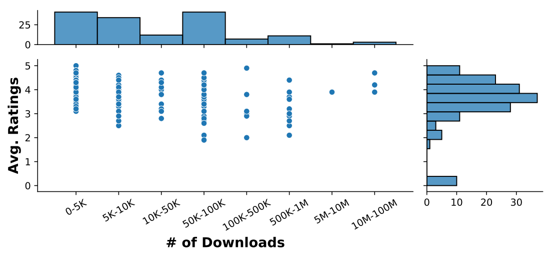
OTP Delivery Methods. Typically, an OTP is a 4- to 8-digit numerical code delivered to the user through remote or token-based methods. In remote delivery methods, the OTP code is generated on a *server* and then sent to the user by SMS, email, or voice. These methods are prone to interception and vulnerable to SIM swap attacks and mobile malware [11,20]. To mitigate these risks, token-based OTP uses special hardware or software at the user end, such as on a computer, smartphone, or smartwatch to generate tokens *locally*.

OTP Generation Protocols. OTP apps use the two commonly employed algorithms for generating OTPs: HOTP (the Hash-based Message Authentication (HMAC) OTP) [45] and TOTP (the Time-based OTP) [46]. The synchronization between the endpoint OTP generator (i.e., OTP app) and server is maintained through the utilization of counter values on both sides in the case of HOTP, and via the use of time stamps (i.e., Unix epoch time) and counter values based on elapsed time increments in TOTP.

OTP Usage. Recent studies on MFA user experiences [10,7] indicates that SMS/Email OTP code delivery was the most popular method, used by 86% of participants. Nevertheless, the surveys also reveal a declining trend in all remote delivery methods and hardware token systems. Conversely,

Fig. 1: (a) OTP app distribution by downloads and ratings. (b) Google Play's top 5 OTP apps by installs and bottom 5 by ratings.

(a)



(b)

No	App Title	# of Installs	Avg. Rating
1	Microsoft Authenticator	50M+	4.7
2	Google Authenticator	50M+	3.9
3	Yono Lite SBI - Mobile Banking	50M+	4.2
4	Twilio Authy Authenticator	10M+	3.9
5	SQUARE ENIX Software Token	1M+	2.9
1	CQUR M-OTP	10+	0.0
2	OTP Friuli Venezia Giulia	100+	0.0
3	NOVOLOAD OTP	100+	0.0
4	UV1.0 OTP	5,000+	0.0
5	BN Token Celular	100,000+	0.0

there is an increasing trend in the usage of OTP codes generated through mobile apps. This trend is reasonable considering that smartphones offer greater usability due to their constant presence with users and are equipped with sensors and cameras for tasks like obtaining biometrics or scanning QR codes effortlessly.

Discovering OTP Apps. We employ a methodology for gathering OTP apps from Google Play and performing their vulnerability analysis. As the Google Play Store does not have a designated category for OTP apps, we developed a custom crawling methodology to identify and collect OTP apps from the Google Play Store. We start by leveraging the search functionality of Google Play with keywords related to OTP such as "one-time password", "OTP", "multi-factor authentication", and "MFA". Next, we utilize the Google Play Store Scrapper [62], a Python library designed for extracting and analyzing app metadata from the Google Play Store, to recursively collect app IDs of *similar* candidates. This method resulted in 629 app IDs.

Next, we collect metadata from OTP apps using Google Play Store Scraper. The collected data included the name, average rating, user reviews, descriptions, and categories of the apps. We manually verified each app's description for accuracy and only selected those that strictly served as OTP applications. Our final dataset consisted of 182 OTP apps, all categorized under *free* applications. We then use `gplaycli` [42] to download their application packages (APK).

Figure 1a shows the distribution of the number of installs and an average rating of these 182 apps. Table 1b (embedded in Fig. 1) shows the top and bottom 5 apps based on the number of downloads and user ratings. Three OTP apps (1.64%) have been installed on more than 50M devices: Google Authenticator, Microsoft Authenticator, and Yono Lite SBI (from the State Bank of India). These apps have an average rating of 4.7, 3.9, and 4.2 respectively, which are regarded as favorable. In contrast, 52 (28.57%) OTP apps with at most 5K installs of which 29 (15.93%) apps were recorded to have an average rating of less than 3 to unrated indicating *negative* users' reviews.

3 Analysis Methodology

Our analysis methodology of assessing the security and privacy of OTP apps involves a custom test suites. This includes source code analysis (§3.1), behavioral analysis of network activity (§3.2), and compliance with privacy policies and user perception (§3.3). The following sections detail the procedures for these analyses.

3.1 Static Analysis

App signing ensures app authenticity via digital signatures. Despite its importance, vulnerabilities like the Janus vulnerability (CVE-2017-13156) [31] allow attackers to modify APKs without breaking their original signatures, affecting apps signed with the V1 Signing Scheme on Android 5.0 to 8.0 [47]. Other issues include Android Master Key vulnerabilities [27] and the misuse of legitimate code signing certificates [61]. We analyze app signing mechanisms investigating the strength of the hash and encryption algorithm, the signing mechanism's version, and the signing's coverage. Using APKSigner [17], we extract information from `CERT.(RSA|DSA|EC)`, `META-INF/MANIFEST.MF`, and `CERT.SF` files.

Android's auto-backup mechanism [1] backs up app data, including sensitive information, to Google Drive. Developers can opt-out by setting `android:allowBackup="false"` in `AndroidManifest.xml`. We assess whether OTP apps permit auto-backup by checking this setting in their manifest files. Unprotected Android components (Activity, Service, Provider, Broadcast Receiver) can be exploited [43]. Tools such as Android Drozer target these components [26], listed in Common Weakness Enumeration [16]. We develop a script to scan `AndroidManifest.xml` for unprotected components, handling custom package names and Activity Alias [29].

As Android manages data exchange through APIs and system calls, we use Androguard [8] to extract and analyze API / system calls to assess security measures such as root detection, biometric 2FA, and adoption of hardware security modules. Next, we inspect decompressed source code of the OTP apps for the presence of any third-party libraries. We identify their presence using a curated list of advertising and tracking libraries [52,33,34]. Due to obfuscation [18,32], our results represent a lower bound. Finally, to detect the presence of malware in the OTP applications, we use VirusTotal[58]. We automate our analysis with VirusTotal's Report API and retrieve malware detection results, identifying any malware signatures.

3.2 Runtime Network Behavior Analysis

To test the resilience of OTP apps in detecting rooted devices, we first run each OTP app in our dataset on a rooted Samsung Galaxy A8+ phone. Using ADB tool [30], we automate installation and invocation of app functionalities to determine if they detect the rooted environment. Next, we analyze the robustness of OTP apps against traffic interception attacks. To this end, we decompile and modify OTP apps to disable SSL Pinning or Certificate Transparency, then test them on a Huawei GR5 phone. Using MITMProxy [44], we intercept traffic to evaluate resilience against interception attacks. Apps that prevent traffic capture are considered resilient. We analyze intercepted traffic to identify data sharing with third-party servers. Compiling a list of first-party domains from Google Play, we filter out these domains to detect third-party access. We cross-validate captured traffic's URLs and second-level domains (SLDs) with EasyList and Easy Privacy filters [25], blocking ads and tracking scripts.

3.3 Compliance and user perception analysis

Evaluating OTP App Developers' Compliance with Privacy Policies. We evaluate compliance by verifying the presence and accessibility of a valid privacy policy link and adherence to stated privacy policies regarding user information sharing. Non-compliance includes missing, faulty, or inaccessible links, and sharing user information contrary to the policy. We developed a tool to automate this analysis, retrieving and extracting privacy policy descriptions and using a machine learning model [63] to classify whether the policy mentions data collection.

Analyzing User Perception about OTP Apps. User reviews are key to evaluating OTP apps. We analyze Google Play comments to understand OTP app users' perceptions. We create scripts to aggregate user reviews and classify them by star rating. We manually examine negative reviews and categorize them into six complaint categories (see Table 7). We focus on keywords relevant to each category. For example, "fake" and "scam" map to the fraud category. We discard comments that generated disagreement in the manual labeling process.

4 OTP Apps Analysis and Results

In this section, we analyze the source code of each OTP app using static analysis techniques as detailed in § 3.1 and § 3.2. Our analysis addresses key research questions on the use of secure encryption and hashing algorithms, certificate signing mechanisms, auto-backup features, vulnerable exported components, and the presence of third-party libraries and potential malware. The following subsections detail our methods and findings.

4.1 Key Size and Signing Mechanism Vulnerabilities in OTP Apps

Do OTP apps use strong enough encryption and hashing algorithms? The integrity and accountability of apps during distribution depend heavily on robust encryption and hashing mechanisms. These mechanisms aim to detect any changes to the app's package, preventing malware insertion.

As shown in Table 1 (Top), we found that OTP apps commonly use RSA/DSA with SHA256 for their digital signatures. Although this combination is generally considered strong, we identified 5 (2.7%) OTP apps using 1024-bit DSA keys and 40 (22.0%) apps using 1024-bit RSA keys, both of which are weak. According to the US Government's standards for Policy and Security Planning, the minimum requirement for public key length is 2048 bits [15]. Among the apps with 1024-bit keys, 8 OTP apps, such as Twilio Authy [12] and Yandex Key [6], were installed over 1 million times, 10 had over 100,000 installs, and the rest had fewer than 50,000 installs.

Table 1: Certificate signing mechanism analysis. Top: Hash and encryption algorithm, Bottom: Versions of signature mechanisms.

Certificate Signing Algorithm

Encryption+ Hash	Key Size	#of Apps(%)	#of Popular Apps (%)	Ex. OTP App (# Downloads)
DSA+SHA256	1024	5(2.8%)	3(1.7%)	Yandex (1M+)
RSA+SHA256	1024	40(22.0%)	16(8.8%)	MOTP (1M+)
RSA+SHA256	2048	79(43.4%)	30(16.5%)	Yono Lite (50M+)
RSA+SHA256	4096	58(31.9%)	11(6.0%)	Microsoft (50M+)

Signature Mechanism Version

Vers. 1	Vers. 2	Vers. 3	#of Apps(%)	#of Popular Apps(%)	Ex. OTP App (# Downloads)
✓	✗	✗	51 (28.0%)	19(10.4%)	BM Soft (500K+)
✓	✓	✗	73 (40.1%)	23(12.6%)	VIB Smart (1M+)
✓	✗	✓	2 (1.1%)	1(0.6%)	Vietcombank(500K+)
✓	✓	✓	55 (30.2%)	17(9.3%)	CIB Corp (50M+)

Do OTP apps have outdated certificate signing mechanisms vulnerable to attack? Table 1 shows the certificate signing mechanisms used by OTP apps. In the analyzed OTP apps, we identified four versions of the certificate signing mechanisms in Android: V1 (since Android 1.0), V2 (since Android 7.0), V3 (since Android 9.0), and V4 (since Android 11.0). V1 is based on JAR signing and has weaknesses addressed in later versions. Google recommends using a tiered mechanism from V1 to the latest version to ensure backward compatibility. As mentioned in Section §3.1, our focus is on identifying OTP apps using the V1 mechanism, which has several vulnerabilities, including CVE-2017-13764 and CVE-2015-1533 [31,47,27]

Do OTP apps' secure cryptographic modules comply with security standards? The Android Keystore system lets the apps store cryptographic keys in a container to make them more difficult to extract from the device. Table 2 shows that 107 (58. 8%) OTP apps used the Keystore class to secure their cryptographic operations, leaving the remaining 41.2% of the apps to treat the cryptographic key as a regular file, making them vulnerable to row hammer attacks [21]. Among the 107, only 18 (9.9%) OTP apps utilized the hardware security module to secure cryptographic processes in their applications, as indicated by the use of `KeyInfo.isInsideSecureHardware()` and `isStrongBoxBacked()` API calls. These hardware security modules refer to a secure-isolated

Table 2: A breakdown of API/System calls utilized by OTP apps based on the security parameters and mechanisms employed.

Security Parameter	Mechanism	API/System Call	# of Apps (%)	# of Pop. Apps	Ex. OTP App (# Downloads)
Rooted device detection	Execute runtime Command (e.g "`su`")	Runtime.exec()	72 (39.6 %)	35 (19.2%)	MOTP-Mobilians(1M+)
	Google Safetynet Attestation	SafetyNet.getClient()	7 (3.8 %)	5 (2.8%)	PacificID(500K+)
	Third-party library	RootBeer.isRooted()	11 (6.0 %)	5 (2.8%)	Entrust Identity(1M+)
		RootTools()	3 (1.7 %)	N/A	Trùm OTP(10K+)
Biometric 2FA	Invoke Fingerprint module	FingerprintManager()	132 (72.5 %)	53 (29.1%)	BM Soft Token(500K+)
	Invoke Biometric module	BiometricPrompt()	23 (12.6 %)	8 (4.4%)	Aruba OTP(500K+)
		BiometricManager()	20 (11.0 %)	8 (4.4%)	QIB Mobile(100K+)
Cryptographic Module	Invoke Keystore system	KeyStore.getInstance()	107 (58.8 %)	45 (24.7%)	Twilio Authy(10M+)
	Hardware Security Module	KeyInfo.isInsideSecureHardware()	17 (9.4 %)	9 (5.0 %)	SQUARE ENIX(1M+)
		isStrongBoxBacked()	1 (0.5 %)	N/A	Eximbank Smart OTP(10K+)

environment or Trusted Execution Environment (TEE), which has various implementations such as Knox Vault [2] in Samsung or Titan M Chip[5] in Google Pixel Firmware. In addition to secure storage, this module has an independent CPU and a true random number generator (TRNG). TRNGs are vital for maintaining password randomness. Research in [40] found that 19.7% of OTP apps that used software-based pseudo-random number generators (PNRGs) failed to comply with RFC 4086 and RFC 4226 randomness requirements for security and the HOTP algorithm.

Takeaway 1. More than 35% of the OTP apps, including many popular ones, use smaller-than-recommended key sizes (1024 bits) for signing. More than 38% of the OTP applications, including popular ones, use V1 of the Android signing mechanism, which is known to have vulnerabilities.

4.2 Vulnerable exported components of OTP apps

Do OTP apps allow auto-backup features? We observed that 92 (50.5%) of the OTP apps in our analysis explicitly disable the auto-backup feature. For the remaining 10 apps, we manually initiated the Android Auto Backup (AAB) functionality and monitored the restoration of data following the uninstallation and reinstallation of each app. Surprisingly, 73 (40.1%) of the analyzed OTP apps utilized the Auto-backup functionality. This represents a significant security risk, as an attacker with access to the user's Google account could potentially access the OTP backup and generate valid OTPs for the user's associated accounts.

Takeaway 2. A notable percentage, 73 (40.1%), of the analyzed OTP apps have enabled the auto-backup feature. Despite encryption measures, device authentication data could be targeted by a throttled attacker [41] to gain access to auto-backup data.

What is the prevalence of improper exported components in OTP apps? We identified improper exported components (Activity, Service, Receiver, and Provider) in OTP apps based on CWE-926 [16], which refers to Android components that can be accessed by any application installed on the device. The code listing 1.1 shows a code snippet of an incorrect exported service in Any OTP Apps (`com.dreammirae.otp.android.mirae.multiid.market`), downloaded by 100K+ users. The code sets `android:exported="true"` without specifying permissions, indicating that the service can be accessed by any application on the device. Such vulnerabilities allow attackers to gain access to sensitive information, modify the internal state of the application, or trick users into interacting with the malicious app.

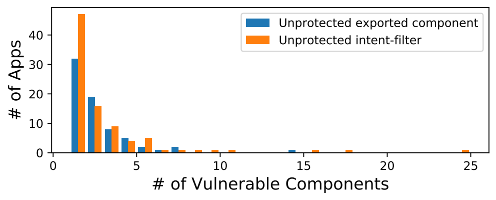
Listing 1.1: Unprotected exported component code snippet

```
<service android:exported="true" android:name="*.RemoteService"/>
```

Another way to expose app's components is by using `Intent-filter`. However, this sets the component exported status to "true" [19]. The improper use of `Intent-filter` in apps such as ACLEDA Bank Security Token (`com.mollatech.android.apps.authenticators`). Such activity can be triggered by any malicious app to manipulate the login activity and gain access to the system. As shown in Figure 2a, 70 (38.5%) OTP apps contained 158 unprotected exported components, while 89 (48.9%) OTP apps had 244 unprotected Intent filters in their `manifest` files.

Fig. 2: Distribution of unprotected components in OTP apps grouped by vulnerability and component type.

(a) Dist. of Unprotected Components in Apps



(b) OTP Apps Vulnerabilities

Vulnerability	Component Type	# of Components	# of Apps(%)	# of Popular Apps(%)
Unprotected Exported Component	Service	81	46(25.3%)	25(13.7%)
	Activity	61	34(18.7%)	16(8.8%)
	Receiver	13	11(6.0%)	7(3.9%)
	Provider	3	3(1.7%)	2(1.1%)
Unprotected Intent-filter	Activity	162	63(34.6%)	26(14.3%)
	Service	47	34(18.7%)	19(10.4%)
	Receiver	32	22(12.1%)	9(5.0%)
	Activity Alias	3	2(1.1%)	1(0.6%)

Next, we analyze the distribution of unprotected components in OTP apps based on the component type. Table 2b shows that `Service` dominated the unprotected exported components with 81 components in 46 (25.3%) OTP apps. Meanwhile, `Activity` dominated the unprotected intent filters with 162 components in 63 (34.0%) OTP apps.

Takeaway 3. We found 46 (25.3%) OTP apps contain unprotected exported components, and 63 (34.0%) apps contain unprotected intent filters in their manifest. This allows malicious applications to access sensitive information, modify the internal state of the application, or trick users into interacting with the malicious application.

4.3 OTP app resource security via APIs and system calls

Do OTP apps detect Jailbroken/rooted devices? Rooted devices grant unrestricted access to apps' data and resources, posing significant security risks [38]. To determine the ability of OTP apps to detect rooted devices, we followed previous studies [57,24,35] by detecting specific API/System calls. We filtered out general API/System Calls, such as `packageManager.getPackageInfo()`, `Os.access()` and `os.stat()` as they are not specifically intended for rooted device detection. Instead, we focused on calls from third-party libraries like RootBeer [3] or RootTools [4], such as `RootBeer.isRooted()` and `RootTools()`. We also scanned the analyzed OPT apps for Google's SafetyNet Attestation API by detecting calls such as `SafetyNetApi.Attestation.Response()` and `SafetyNet.getClient()` [9].

Our analysis, presented in Table 2, revealed that 72 (39.6%) OTP apps used `Runtime.exec()` to access system calls and execute kernel runtime commands. 7 (3.8%) OTP apps leveraged the official Google SafetyNet Attestation. This number is lower compared to the 11 (6.0%) apps relying on third-party source code like Root Beer. Among the 75 (41.2%) OTP apps with Jailbreak/root detection mechanisms, 2 (1.1%) used four strategies, 4 (2.2%) used three mechanisms, 14 (7.7%) used two mechanisms, and 55 (30.2%) used one mechanism.

Takeaway 4. While most OTP applications use protections such as rooted device detection, a significant number use deprecated APIs for biometric authentication, and they *use Android Keystore for cryptographic keys with non-compliant pseudo-random number generators, leaving OTP apps vulnerable to replay attacks.*

4.4 Presence of Third-party Libraries and Malware

Do OTP apps rely on third-party libraries? Third-party libraries play a crucial role in an app's ability to share information with external entities. We analyze four different types of libraries that share user information with third parties. These include Ads & Tracker, Social Media Networking, Payment, and Analytics. Based on this criterion, we found 6 different libraries (from our list of 383 libraries mentioned in Section §3.1) embedded in 73 (40.1%) OTP apps. The results, presented in Table 3, indicate Google Ads dominate other libraries, being embedded in 54 (29.7%) OTP apps. On the other hand, 59.9% (109) of the OTP apps do not include any third-party libraries in their code.

Only 2 OTP apps, accounting for 1.1% of the total, adopt multiple libraries: Microsoft Authenticator embeds Google Ads, Adjust, Facebook, and Squareup, while OTP SmartBank includes Facebook, Google Ads, Google Analytics, and Squareup in its source code.

Table 3: Top 3rd-party libraries distributions in OTP apps.

Lib. Names	Lib. Type	# of Apps (%)	# of Pop. Apps (%)
Google Ads	Ads & Tracker	54 (29.7%)	26(14.3%)
Squareup	Payment	24 (13.2%)	12(6.6%)
Facebook	Social Media	14 (7.7%)	5(2.7%)
Google Analytics	Analytics	7 (3.8%)	5(2.7%)
Adjust	Analytics	1 (0.5%)	1(0.5%)
Tencent	Ads & Tracker	1 (0.5%)	N/A

This adoption of third-party libraries is lower compared to other genres of apps. For example, 22.2% of Android health-related apps embed at least 5 different third-party libraries found by Sentana et al. [53], while 43% of non-health apps include more than 5 ads & tracking libraries, as reported in [55]. This lower rate of adoption is due to the nature of the OTP app business. Most of these OTP apps in our corpus are affiliated with reputable enterprises in the banking or IT sector, thus third-party libraries are not the primary source for monetizing these businesses. Also, the security requirements of OTP necessitate a limited use of third-party libraries.

Takeaway 5. More than 40% of OTP apps include third-party libraries. While this is lower than non-OTP apps, it is surprisingly high as sharing data with third parties through OTP apps is not the primary monetization source for OTP service providers.

Do OTP apps contain suspicious codes? VirusTotal results show that 3 (2%) OTP apps contain malware in their source codes. The SWIFT Token [54], with more than 1,000 downloads, was detected with malicious code by Avira and Cynet anti-virus engines. Avira identified `ANDROID/Agent.FJMK.Gen`, and Cynet flagged the code with a score of 99. Yandex Key [6] and the BC OTP app [13], both with more than 1 million downloads, were also found to contain malicious code by the MaxSecure anti-virus engine, which detected `Android.WIN32.Boogr.gsh` malware. Fortiguard classifies this malware as a trojan, known for capturing keyboard input, gathering system information, and spreading other malware [36].

Next, we investigate the reasons why these OTP applications are labeled suspicious by VirusTotal. To identify potential suspicious APIs or code, we scan third-party libraries embedded in OTP apps and compare them to a list of suspicious libraries [60]. Our analysis revealed that one of the OTP apps analyzed engages in malicious practices by embedding Cross-Library Data Harvesting (XLDH) libraries. XLDH libraries steal user information from legitimate libraries such as Facebook, Google, Twitter, and Dropbox. Facebook has taken legal action against several companies providing such libraries [60]. Specifically, Yandex Key [6] embeds the `com.yandex.metrica` XLDH library. In the following, Our dynamic analysis, confirmed that this library actively collects Google Advertising ID, Android ID, and user MAC address, exfiltrating the information to `yandex.net` via HTTPS flows. Yandex Key is an independent OTP authenticator, highly rated with an average rating of 3.7, and installed on over 1M devices.

Takeaway 6. OTP apps have stricter security requirements than other genres of apps but still rely on third parties and are susceptible to malware attacks. Although most OTP apps are malware-free, as one would hope, we found 3 OTP apps that contain some form of malware.

Are OTP apps resilient against traffic interception? Two common methods for protecting network traffic against Man-in-the-Middle attacks in Android are Certificate/SSL Pinning [59] and Certificate Transparency [23]. Identifying these mechanisms via static analysis is challenging due to numerous libraries [37,49] and variable implementations. Following previous studies [22,23], we used MITMproxy's `ssl-insecure` mode to analyze OTP apps' defenses. When encountering an untrusted certificate, MITMproxy failed to capture network traffic, returning a consistent exception message for both SSL Pinning and Certificate Transparency

Next, we downgrade the security of OTP apps in order to force the apps to trust the connection to the proxy server using MITMproxy. We create a custom or self-signed CA and added it to the proxy server's trusted Keystore. We decompile the OTP apps using APKtool and modify the `network_security_config.xml` file to direct the certificate searching to the proxy server's Keystore. This modification strategy forces the apps to trust any credentials available in the proxy server. We also modify the `Manifest.xml` to direct `android:networkSecurityConfig` to the modified `network_security_config.xml`. We repackage the hacked apps using APKtool and assign them a self-signed certificate similar to the stored certificate in the trusted Keystore. This is done using JAR-signer (app's signing mechanism, cf. §3). We build a testbed, to test each modified OTP app. To reduce noise from other installed and prepacked apps on our testing mobile phone, we block background traffic for all apps except the OTP app under test and navigate the apps manually.

Table 4: Top: Result of OTP apps against Network Interception Attacks. Bottom: Cross Validation result of OTP apps failed to repackage to Anti-analysis result. ✓represents the adoption, A.Ds: Anti Disassembly, A.Db.: Anti Debug, Pck.: Packer and Obf.: Obfuscator.

OTP Apps Status	**# of Apps (%)**	**Example App ()**
Resilience Against Interception Attacks	119 (65.4%)	Google (50M+)
Vulnerable to Interception Attacks	58 (31.9%)	Lenovo(10K+)
Failed to Repackage	5 (2.8%)	

App Name	**# of Down.**	**A.Ds.**	**A.Db.**	**Pck.**	**Obf.**
Aircuve Mobile	5K+	✓			✓
Hangame OTP	10K+			✓	
Namirial OTP	100K+	✓	✓		✓
BIDV Smart OTP	100K+	✓	✓		✓
ALEXBANK Key	100K+		✓		

Table 4 shows the results of our analysis. We found that 119 (65.4%) OTP apps detected traffic interceptions. However, a significant number of OTP apps 58 (31.9%) failed to detect the interception. For example highly popular apps OTP mobile banking HR and OTP SmartBank, both have over 100K installs, are vulnerable to network traffic interception attacks. 5 (2.7%) OTP apps failed our repackaging attempts. We cross-validate these 5 apps to those that use binary protection (§3). The bottom half of Table 4, highlights the cross-validation between these 5 apps and the anti-analysis technique adoption including Anti Disassembly, Anti Debug, Packer, and Obfuscator. Our results demonstrate that it is challenging to intercept the network traffic of OTP apps that use anti-analysis techniques.

Takeaway 7. A significant portion (31.9%) of OTP apps are vulnerable to network traffic interception, despite the majority being able to cope with it. Intercepting the network traffic of OTP apps that have adopted anti-analysis techniques presents a challenge.

Do OTP apps communicate data with third-party tracking and analytic services? Using the network traffic interception results, detailed above, we extract the network traffic dump into an HTTP archive file (HAR) and then leverage the `TLD python` library to extract the `second level domain` (SLD) of each network request that we manage to capture. We remove the traffic that directs to the first-party server based on the SLD of the developer's website extracted during metadata crawling (§2). Then, we compare the SLD to Easylist and EasyPrivacy which contains a list of domains, and subdomains that are used to block domains that facilitate or provide advertising content. In Table 5, we found that 24 (13.2%) OTP apps had at least one SLD belonging to ad domains from EasyList. Similarly, we observed that 26 (14.3%) OTP apps had at least one of their SLDs labeled as a tracking domain from the EasyPrivacy list.

Table 5: Third-party domains requested by OTP apps.

Domain Level	Reference List	# of Apps (%)	# of Popular Apps (%)	App Name (# Downloads)
Second Level	EasyList	24(13.2%)	10(5.5%)	VietCB(500K+)
Domain (SLD)	EasyPrivacy	26(14.3%)	10(5.5%)	Código C.(1M+)
Netloc	Easy List	17(9.3 %)	4(2.2%)	MToken (50K+)
	Easy Privacy	5(2.8%)	1(0.5%)	SGuard(5K+)

Takeaway 8. The presence of third-party ads and tracking domains in the network traffic suggests that a significant fraction (13.2%) of OTP apps may potentially leak information to third-parties.

5 Compliance and Users Perception

The previous sections have identified instances of OTP apps with vulnerable components, malware presence, and various security and privacy risks. In this section, we take both developer-centric and user-centric perspectives to assess compliance with privacy policies and examine whether users publicly report any privacy and security issues related to OTP apps in their Google Play reviews.

OTP apps' privacy compliance? We run our crawler to traverse the privacy policy link provided by the OTP app developers on the Google Play store and extract the privacy policy text. As a result, we managed to crawl the privacy policy text of 115 (63.2%) apps but failed to obtain the corresponding text from the remaining 67 (36.8%) OTP apps due to several reasons.

Lack of privacy policy disclosure. We then manually checked the privacy policy link's existence and traversed the links that failed to be crawled by our script. We found 32 (17.6%) OTP apps did not provide privacy policy links or did not provide metadata related to the privacy policy on the Google Play Store page and then marked them as non-compliant apps. We then excluded 17 OTP apps that return "connection reset by host", "CERTIFICATE_VERIFY_FAILED", or " temporary failure in the name resolution" errors from the non-compliance list because these errors are a consequence of defense mechanism against our crawler. In general, as listed in Table 6, we found that 50 (27. 47%) of the OTP apps in our corpus failed to provide a web page to disclose their privacy policy to the user. Surprisingly, highly rated and popular apps such as ActivID Token and Mobile-OTP (each has more than 100K downloads and avg. rating of $\geq$ 3.5) do not provide any link on the Google Play store.

Table 6: The cause of the failure of Privacy Policy extraction of OTP apps. ✓indicates manual traverse succeeded and ✗indicates manual traverse failed.

Caused of Failure	# of Apps (%)	Manual Traverse
No Privacy Policy Link or No Metadata	32 (17.6%)	✗
Temporary failure in name resolution	11 (6.0%)	✓
HTTP Error 404: Not Found	9 (4.9%)	✗
Connection reset by peer	3 (1.6%)	✓
CERTIFICATE_VERIFY_FAILED	3 (1.6%)	✓
Name or service not known	2 (1.1%)	✗
HTTP server returned an infinite loop.	2 (1.1%)	✗
Internal Server Error	2 (1.1%)	✗
No address associated with hostname	1 (0.2%)	✗
Connection timed out	1 (0.5%)	✗
No route to host	1 (0.5%)	✗
Failed to Crawl	**67(36.8%)**	
Failed to Disclose Privacy Policy		**50(27.47%)**

Adherence to information sharing policy. We analyze if OTP apps adhere to their user information sharing policies in two steps (as shown in Figure 3): (*i*) classifying if an app's privacy policy states that it shares user information with third parties (TPs) or not, and (*ii*) validating if the information is indeed not shared with TPs if so claimed. *Firstly*, for each app, we classify its privacy policy as *true* or *positive* if it declares that the app shares user information with TPs. Otherwise, it is labeled

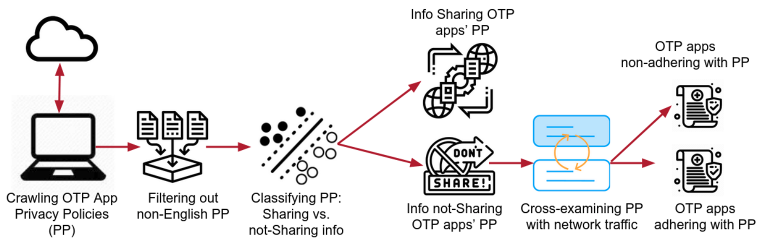

Crawling OTP App Privacy Policies (PP)
Filtering out non-English PP
Classifying PP: Sharing vs. not-Sharing info
Info Sharing OTP apps' PP
Info not-Sharing OTP apps' PP
DON'T SHARE!
Cross-examining PP with network traffic
OTP apps non-adhering with PP
OTP apps adhering with PP

Fig. 3: Compliance analysis pipeline for OTP Apps: Our process involves identifying apps that disclose and explicitly outline their data-sharing practices in their privacy policies (PP). Additionally, we identify instances where apps deviate from their stated privacy policies, indicating that they claim not to collect or share information but engage in such practices, as detected through network traffic analysis.

false or *negative*. *Secondly*, we cross-validate the apps whose privacy policies are labeled as false by checking the third-party domains accessed during the app's runtime. We assume that an OTP app violates its privacy policy in terms of sharing user information with TPs if they do not declare it in privacy policies while simultaneously embedding tracking libraries or sending user information to third-party domains.

We use an annotated corpus of privacy policies [63] to detect if the app allegedly shares user information with third parties. The corpus, annotated by legal experts, has 213 policies labeled positive and 137 negatives for information sharing. However, the positive annotation only applies to specific phrases like "We share your email address with TPs" and general phrases are labeled negative. To correct this, we re-label the corpus by removing general information sharing from the negative class. We reduce the negative subset to 35 policies to balance class data and create a new corpus with 68 positive and 35 negative policies.

In the same spirit of previous works such as [63], we use `scikit-sklearn` to train a Support Vector Machine (SVM) classification model using this corpus. We exclude 48 (out of 115) policies as they were not written in English, including binary text. We then classify the remaining 67 policies using our trained model. Results showed 30 policies classified as False (no declaration of user information sharing with TPs) and 37 as True. We cross-validated the 30 OTP apps whose privacy policies were labeled as False, against the presence of ads and tracking libraries as discussed in Section §4. Our results showed that 12 (6.6%) OTP apps, such as Twilio Authy (with 10M+ downloads and 3.8 average ratings) and STOVE Authenticator (with 10K+ downloads and 4 average ratings) fell into this category, implying they failed to adhere to their privacy policies regarding sharing of user information with third parties.

Takeaway 10. At least 12 (6.6%) OTP apps failed to comply with their privacy policies in terms of not declaring that they share user information with third parties.

How do users perceive OTP apps? Given the lack of standards or guarantees for security and privacy for users of OTP apps besides the reputation of developers, user reviews need to be taken into account as an indicator in choosing OTP apps. In general, we found 251,657 negative reviews relating to 147 of 182 OTP apps in our corpus. The remaining 19.8% of OTP apps such as "VIB Smart OTP" and "PV Smart OTP" either had no negative reviews or received no reviews from users. In our labeling process for classifying reviews into categories (cf. §3.3), we employ a majority voting scheme to resolve any disparities. Out of the total of 251,657 negative reviews, a total of 2,984 user comments, accounting for 1.2% of the total, were discarded due to disagreement.

These 248,673 negative reviews are categorized based on the type of complaint: fraud, security, authentication, bugs, usability, and ads & tracker. Table 7 shows that most user complaints are related to the authentication mechanism which is recorded in 28. 4% of the total negative review and distributed in 64 (32. 5%) OTP applications. In comparison, 72 (39. 6%) of the OTP apps were reported

Table 7: Overview of categories of users' complaints about OTP apps.

Category	**# of Complaint(%)**	**#of Apps(%)**	**#of Popular) Apps(%)**	**Ex. OTP App # Downloads**	**Keywords**
Authentication	15,151 (28.4%)	64 (35.2%)	64(35.2%)	Yono Lite SBI(50M+)	*verification, verify, account, login, notification, register*
Bugs	9,909 (18.6%)	72 (39.6%)	15(8.2%)	SBI Secure OTP(1M+)	*bug, error, crash, update, stuck, upgrade, freeze, not responding*
Usability	3,083 (5.8%)	60 (33.0%)	60(33.0%)	Twilio Authy(10M+)	*confuses, confusing, bad, slow, rubbish, junk, user interface*
Security	2,169 (4.1%)	41 (22.5%)	41(22.5%)	Entrust Identity(1M+)	*security, (in)secure, bot, hacking*
Fraudulent	157 (0.3%)	19 (10.4%)	19(10.4%)	CanPass(100K+)	*scam, fake, money, liar, manipulation*
Ads and Tracker	127 (0.2%)	19 (10.4%)	19(10.4%)	Yubiko (100K+)	*ads, video ads, tracker, advertisement, intrusive ads, massive ads*

to contain bugs with a total complaint of 9,909 or 18.9% of negative reviews. We also highlight the unusual fact of 127 complaints related to advertisements and trackers distributed across 19 (10.4%) OTP apps. We then compare this complaint to the third-party library adoption of the OTP apps in §4 and found all of these apps embed Google ads and Facebook ads library in their package.

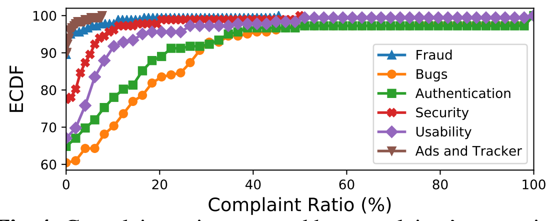


Fig. 4: Complaint ratios grouped by complaints' categories.

To observe the complaint distribution in OTP apps, we measured the ratio of complaints per category to the total negative reviews for each app to account for differences between apps with many and few reviews. For instance, Twilio Authy has a bug complaint ratio of 7.4% calculated from 141 bug complaints divided by 1,898 negative reviews recorded by the app. Figure 4 shows the distribution of the ratio of negative reviews to the total number of reviews across various categories. It reveals that the distribution of bug complaints is the most common category with 72 (over 60%) OTP apps recording complaints, ranging from 5% to 100%. In contrast, fraud was recorded in 19 (less than 15%) apps such as Zenith Ban (100K+ downloads, 2.8 avg. rating) and SQUARE ENIX (1M+ downloads, 2.9 avg. rating), with a complaint ratio below 10%.

Takeaway 11. A surprisingly high number of OTP apps have user complaints about their security (41), the presence of bugs (72), accusations of being fake (19), and containing ads and trackers (19).

6 Disclosure

We notified 146 significantly impacted app developers 17 days before submission. We received 14 automated replies and 17 emails failed to deliver due to invalid email addresses. To date, we have responses from 25 (17%) OTP app developers, including automated responses. Entrust responded promptly by publishing a level 3 severity ticket and escalated the problem to their security team. Eset and PinGo Auth set up special channels for further discussion. Yandex Key responded and also escalated the issue to their security teams. Twilio Authy confirmed our findings and informed the vulnerabilities are fixed in the next app version. Two bank-related OTP app developers declined discussions due to confidentiality and security reasons but took our findings seriously. We plan to revise our paper based on responses and communication received from app developers.

7 Related Work

Several studies have analyzed the security of SMS-based 2FA used by popular cloud services like Google, Dropbox, Twitter, and Facebook. Dmitrienko et al. [20] conducted a security assessment of SMS-based mobile two-factor authentication, commonly employed by prominent cloud-based services like Google, Dropbox, Twitter, and Facebook. Their study revealed multiple vulnerabilities that could enable an attacker to circumvent the authentication mechanisms of these services. The study also introduced an attack scenario that leverages cross-platform infection to compromise control over both PC and mobile device endpoints engaged in the authentication protocol.

Concurrently, another study by Tzemoz et al., [56] employed dynamic analysis to assess the effectiveness and intricacy of OTP system security. Their findings indicated a direct correlation between the level of security and the complexity of computation, as well as the architecture of the authentication security support system. Ozkan et al., [50] tested 11 well-known OTP authenticators with static analysis parameters while Pollit et al., [51] conducted a forensic analysis of 16 OTP apps to determine security keys for storage and security parameters. These criteria encompassed aspects such as obfuscation, resistance to tampering or repackaging, secure storage mechanisms, and device binding considerations. Their objective was to ascertain the location of secret key storage and assess security through multiple parameters, including the presence of encrypted keys, PIN protection, and the implementation of SSL connections. Ma et al. [39] analyzed OTP implementation parameters, including OTP randomness, length, retry attempts, and expiration, by testing 544 apps in the Google Play Store and 3,303 apps on the Tencent Marketplace. The study in [40] focuses on standardizing randomness invocations of OTP authenticators.

The closest study to ours is in [28], which examines the key recovery and backup functions of only 22 popular TOTP Android apps. In contrast, we argue that our study is more comprehensive as it performs a detailed OTP app ecosystem on Google Play. Our analysis consists of the identification of potentially vulnerable components as well as the security and privacy features of the OTP apps. Our static analysis covers the resilience of the encryption algorithm, the signing of applications, binary protection, third-party libraries, security features, and API calls. During run-time, behavior analysis tests the resilience of OTP apps against traffic intercept, information leakage, and Jailbreak detection. In addition, the study evaluates the compliance of the developer with the privacy and information sharing policies with third-party servers.

Our results suggest that a significant number of OTP app users may not be aware of the low security and usability standards provided by these apps, even with the promises made by most of them. To promote continued research in this field, we are making our dataset and scripts accessible to the research community at `https://github.com/otpappsanalyzer/otpappsanalyzer`.

8 Discussion and Concluding Remarks

OTP apps are required to have higher security and privacy compared to other apps due to their role in protecting systems by strengthening their authentication mechanism. However, in this study, we found that many OTP apps failed to meet these standards. Here are several concerns as well as recommendations for app developers to improve their app security.

We argue that OTP apps should fulfill minimum security standards to ensure that their user's data is secure. For example, we found 51 OTP apps that use the vulnerable version 1 of the Android app signing mechanism. Likewise, 70 and 89 OTP apps contained unprotected exported components and unprotected intent filters in their code. Moreover, OTP apps should combine multiple security features, such as leveraging hardware-backed cryptographic modules, conducting Jailbreak detection on the running device, and using biometric authentication. All of these precautions can be used to improve the security of the app against privilege-escalation attacks and to ensure that only authorized users can access the system. We recommend that OTP apps use some sort of protection mechanism to make it harder to reverse-engineer their binaries. Most employ virtual machine detection, but less than 30% of the OTP apps adopt other binary protection techniques such as anti-debug and anti-disassembly. Since OTP authenticators are not the primary revenue source for most providers, OTP apps are often part of broader service agreements, such as email or cloud services. Therefore, we recommend that OTP apps avoid using third-party tracking and ad libraries, as these share user information and threaten privacy.

References

1. Back up user data with auto backup. https://web.archive.org/web/20220421053001/https://developer.android.com/guide/topics/data/autobackup (2023)
2. Knox vault. https://docs.samsungknox.com/admin/whitepaper/kpe/knox-vault.htm (2023)
3. Rootbeer. https://github.com/scottyab/rootbeer (2023)
4. Roottools. https://github.com/Stericson/RootTools (2023)
5. Titan security key. https://cloud.google.com/titan-security-key (2023)
6. AG, I.S.: Yandex key. https://play.google.com/store/apps/details?id=ru.yandex.key (2023)
7. Amft, S.e.a.: "we've disabled mfa for you": An evaluation of the security and usability of multi-factor authentication recovery deployments. In: CCS (2023)
8. Androguard: Androguard reverse engineering, malware analysis of android applications ... and more (ninja) ! https://github.com/androguard (2023)
9. Android: Safetynet attestation api. https://developer.android.com/training/safetynet/attestation (2022)
10. Anise, O., Lady, K.: State of the Auth: Experiences and Perceptions of Multi-Factor Authentication. Duo Security Lab Inc. (2017)
11. Ankita, K., Patankar, A.B., Tawde, P.: Sms-based one time password vulnerabilities and safeguarding otp over network. IJERT (2014)
12. Authy: Twilio authy authenticator. https://play.google.com/store/apps/details?id=com.authy.authy (2023)
13. Biscout: Bc otp. https://play.google.com/store/apps/details?id=com.atsolution.android.bcotp (2023)
14. Brian, K.: The rise of one-time password interception bots. https://krebsonsecurity.com/2021/09/the-rise-of-one-time-password-interception-bots/ (2021)
15. CSRC: Recommendation for key management, part 1: General (revision 3), computer security resource center. https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-3/archive/2012-07-10 (2016)
16. CWE-926: Cwe-926: Improper export of android application components. https://cwe.mitre.org/data/definitions/926.html (2024)
17. Developer, A.: Command line tootls - apksigner. https://developer.android.com/studio/command-line/apksigner (2021)
18. Developer, A.S.: Shrink, obfuscate, and optimize your app. https://developer.android.com/studio/build/shrink-code (2020)
19. Developers, A.: Android documentation - intent and intent filter. https://developer.android.com/guide/components/intents-filters (2022)
20. Dmitrienko, A., Liebchen, C., Rossow, C., Sadeghi, A.R.: On the (in)security of mobile two-factor authentication. In: Christin, N., Safavi-Naini, R. (eds.) FCDS (2014)
21. Documentation, A.: Hardware security best practices. https://source.android.com/docs/security/best-practices/hardware/ (2022)
22. Dolan, M.: Android security: Ssl pinning. https://appmattus.medium.com/android-security-ssl-pinning-1db8acb6621e (2017)
23. Dolan, M.: Android security: Certificate transparency. https://appmattus.medium.com/android-security-certificate-transparency-601c18157c44 (2019)
24. Druffel, A., Heid, K.: Davinci: Android app analysis beyond frida via dynamic system call instrumentation. In: ACNS. Cham (2020)
25. Easylist: Easylist and easyprivacy list. https://easylist.to/ (2023)
26. F-Secure-Labs: Drozer: Comprehensive security and attack framework for android. https://labs.f-secure.com/tools/drozer/ (2024)
27. Freeman, J.: Android bug superior to master key. http://www.saurik.com/id/18 (2019)
28. Gilsenan, C., Shakir, F., Alomar, N., Egelman, S.: Security and privacy failures in popular 2FA apps. In: USENIX SEC (2023)

29. Google: activity-alias. https://developer.android.com/guide/topics/manifest/activity-alias-element (2023)
30. Google: Android debug bride (adb). https://developer.android.com/studio/command-line/adb (2023)
31. Guardsquare: New android vulnerability allows attackers to modify apps without affecting their signatures. https://www.guardsquare.com/blog/new-android-vulnerability-allows-attackers-to-modify-apps-without-affecting-their-signatures-guardsquare (2019)
32. Guardsquare-Mobile-Application-Protection: Dexguard: Android app security - protecting android applications and sdks against reverse engineering and hacking. https://www.guardsquare.com/en/products/dexguard (2020)
33. Ikram, M., Kaafar, M.A.: A first look at mobile ad-blocking apps. In: NCA (2017)
34. Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M.A., Paxson, V.: An analysis of the privacy and security risks of android vpn permission-enabled apps. In: IMC (2016)
35. Kim, T., Ha, H., Choi, S., Jung, J., Chun, B.G.: Breaking ad-hoc runtime integrity protection mechanisms in android financial apps. In: AsiaCCS (2017)
36. Labs, F.: Threat encyclopedia - android/boogr.gsh!tr. https://www.fortiguard.com/encyclopedia/virus/7166894 (2021)
37. Lachimov, A., Verbeeck, N., Dolan, M., Leszczynski, M.: appmattus/certificatetransparency. https://github.com/appmattus/certificatetransparency (2022)
38. Luca Casati, A.V.: The dangers of rooting: Data leakage detection in android applications. Mobile Information Systems (2018)
39. Ma, S., Feng, R., Li, J., Liu, Y., Nepal, S., Diethelm, Bertino, E., Deng, R.H., Ma, Z., Jha, S.: An empirical study of sms one-time password authentication in android apps. In: ACSAC (2019)
40. Ma, S., Li, J., Kim, H., Bertino, E., Nepal, S., Ostry, D., Sun, C.: Fine with "1234"? an analysis of sms one-time password randomness in android apps. In: ICSE (2021)
41. Markert, P., Bailey, D.V., Golla, M., Dürmuth, M., Aviv, A.J.: On the security of smartphone unlock pins. ACM Transactions on Privacy and Security (TOPS) **24**(4), 1–36 (2021)
42. matlink: Google play downloader via command line v3.25. https://github.com/matlink/gplaycli (2018)
43. Melamed, T.: Hacking android apps through exposed components. https://www.linkedin.com/pulse/hacking-android-apps-through-exposed-components-tal-melamed (2020)
44. mitmproxy: mitmproxy - an interactive https proxy. https://mitmproxy.org (2023)
45. M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., Ranen, O.: Request for comments (rfc) 4226 - hotp: An hmac-based one-time password algorithm. https://www.ietf.org/rfc/rfc4226.txt (2005)
46. M'Raihi, D., Machani, S., Pei, M., Rydell, J.: Request for comments (rfc) 6238 - totp: Time-based one-time password algorithm. https://www.rfc-editor.org/rfc/rfc6238 (2011)
47. NIST: National vulnerability database - cve-2017-13156. https://nvd.nist.gov/vuln/detail/CVE-2017-13156 (2019)
48. Ocampo, V.R..: Most mobile authenticator apps have a design flaw that can be hacked. https://www.businesswire.com/news/home/20211008005015/en/Most-Mobile-Authenticator-Apps-Have-a-Design-Flaw-That-Can-Be-Hacked (2021)
49. OkHttp: Certificatepinner. https://square.github.io/okhttp/4.x/okhttp/okhttp3/-certificate-pinner/ (2022)
50. Ozkan, C., Bicakci, K.: Security analysis of mobile authenticator applications. In: ISCTURKEY (2020)
51. Polleit, P., Spreitzenbarth, M.: Defeating the secrets of otp apps. In: 2018 11th International Conference on IT Security Incident Management & IT Forensics (IMF). pp. 76–88 (2018). https://doi.org/10.1109/IMF.2018.00013
52. Seneviratne, S., Kolamunna, H., Seneviratne, A.: A measurement study of tracking in paid mobile applications. In: WiSec (2015)
53. Sentana, I.W.B., Ikram, M., Kaafar, M.A., Berkovsky, S.: Empirical security and privacy analysis of mobile symptom checking apps on google play. In: SECRYPT (2021)
54. i Sprint: Swift token. https://play.google.com/store/apps/details?id=com.isprint.SWIFTToken (2023)

55. Tangari, G., Ikram, M., Sentana, I.W.B., Ijaz, K., Kaafar, M.A., Berkovsky, S.: Analyzing security issues of android mobile health and medical applications. JAMIA **28** (2021)
56. Tzemos, I., Fournaris, A.P., Sklavos, N.: Security and efficiency analysis of one time password techniques. In: PCI (2016)
57. Uddin, M.S., Mannan, M., Youssef, A.: Horus: A security assessment framework for android crypto wallets. In: Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M. (eds.) SSCN (2021)
58. VirusTotal: Anti-virus engines. `https://www.virustotal.com/gui/home/upload` (2024)
59. Walton, J., Steven, J., Manico, J., Wall, K., Iramar, R.: Certificate and public key pinning. `https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning` (2022)
60. Wang, J., Xiao, Y., Wang, X., Nan, Y., Xing, L., Liao, X., Dong, J., Serrano, N., Lu, H., Wang, X., Zhang, Y.: Understanding malicious cross-library data harvesting on android. In: USENIX SEC (2021)
61. Whittaker, Z.: Hackers are selling legitimate code-signing certificates to evade malware detection. `https://www.zdnet.com/article/hackers-are-selling-legitimate-code-signing-certificates-to-evade-malware-detection/` (2018)
62. Yu, J.M.: google-play-scraper. `https://pypi.org/project/google-play-scraper` (2023)
63. Zimmeck, S., Story, P., Smullen, D., Ravichander, A., Wang, Z., Reidenberg, J., Russell, N.C., Sadeh, N.: Maps: Scaling privacy compliance analysis to a million apps. PETS (2019)