Received 28 June 2024, accepted 30 July 2024, date of publication 5 August 2024, date of current version 22 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3439095

RESEARCH ARTICLE

"You Received $100,000 From Johnny": A Mixed-Methods Study on Push Notification Security and Privacy in Android Apps

THOMAS NETELER[1], **SASCHA FAHL**[2], **AND LUIGI LO IACONO**[1]
[1]Department of Computer Science, H-BRS University of Applied Sciences, 53757 Sankt Augustin, Germany
[2]CISPA Helmholtz Center for Information Security, 30159 Hannover, Germany

Corresponding author: Thomas Neteler (thomas.neteler@h-brs.de)

This work was supported in part by Bonn-Rhein-Sieg University of Applied Sciences.

ABSTRACT Push notifications are widely used in Android apps to show users timely and potentially sensitive information outside the apps' regular user interface. Google's default service for sending push notifications, Firebase Cloud Messaging (FCM), provides only transport layer security and does not offer app developers message protection schemes to prevent access or detect modifications by the push notification service provider or other intermediate systems. We present and discuss an in-depth mixed-methods study of push notification message security and privacy in Android apps. We statically analyze a representative set of 100,000 up-to-date and popular Android apps from Google Play to get an overview of push notification usage in the wild. In an in-depth follow-up analysis of 60 apps, we gain detailed insights into the leaked content and what some developers do to protect the messages. We find that (a) about half of the analyzed apps use push notifications, (b) about half of the in-depth analyzed messaging apps do not protect their push notifications, allowing access to sensitive data that jeopardizes users' security and privacy and (c) the means of protection lack a standardized approach, manifesting in various developer-defined encryption schemes, custom protocols, or out-of-band communication methods. Our research highlights gaps in developer-centric security regarding appropriate technologies and supporting measures that researchers and platform providers should address.

INDEX TERMS Push notifications, end-to-end security, android, FCM, intermediate systems.

I. INTRODUCTION

Mobile applications use push notifications to inform their users about time-critical information, like received messages, without them having to open the apps actively. Such contexts require "pushing" data from the app server out to its clients. This requires a permanent connection between the client device and the server, as the server cannot initiate a connection to a client. Every app that wants to use push notifications therefore requires such a permanent connection [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Shuangqing Wei.

In practice, mobile operating systems provide functions for push notifications by establishing and maintaining the connection. This enables notifications even when an app is not running and allows the multiplexing of notifications for multiple apps simultaneously. Moreover, this approach significantly reduces power consumption, as not every app that requires push notifications needs to open and maintain a communication connection, as well as a running background process.

On the other hand, this introduces a third party besides the app server and client apps: the provider of the push notification service. It adds an internal device component (interface for client apps) and an additional server component (interface for app server) to the system. On Android, the Firebase Cloud Messaging (FCM) service from Google is the default for integrating push notifications into apps.[1] FCM works with the Play Services that handle the on-device part and are responsible for connection management and forwarding messages to the corresponding apps.

Android distinguishes between two types of push notifications. "Notification Messages" are processed by the OS and triggered directly without interacting with the corresponding app. "Data Messages" are forwarded by the OS to the corresponding app and are processed there. Notification Messages cannot be protected by end-to-end security mechanisms as the OS needs to access the message content for processing. In contrast, app developers are free to decide whether they want to implement message-oriented security when using Data Messages.

A. THREAT MODEL

FCM only secures the transport between its own servers, the app server, and the client device using TLS (cf. Figure 1). FCM messages are encoded in the ProtoBuf format, which does not contain security measures such as encryption or integrity protection.[2] If the developers do not protect the content of the messages themselves, all push notifications can be read, manipulated, or suppressed by the service provider. Neither the client nor the app vendor's server can prevent or detect the interception, modification, or suppression of notifications without further protective measures for the notification message. Security vulnerabilities, malicious actors, or law enforcement can exploit such a lack of end-to-end security to gain access to push notifications. This poses a serious threat to the security and privacy of app users if the notifications contain sensitive or private information. To achieve *adequate security and privacy* in such cases, app developers must implement message-oriented protection in addition to the transport protection mechanisms implemented by default in the form of TLS.

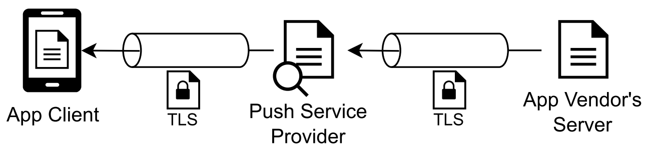


FIGURE 1. A push notification payload is sent from an app server to a client device via a push notification service provider, such as FCM. Although the transport is encrypted, the service provider can still access the content of the push notification.

B. MOTIVATION

Notification providers on top of OS vendor-specific delivery mechanisms (like FCM on Android) were explored previously. They showed data leakage of notification on several fronts: cloud provider console, other apps on the device, and data exchange with such host notification providers not directly visible to the user [2].

Recent discussions about law enforcement access to push notifications and their metadata showed that there is a substantial interest in push notification content. Although the metadata like push tokens or timestamps seems to be the most interesting data, payload data was also requested showing that push notification providers store data about their notifications [3], [4], [5].

Protecting the payload can mitigate leaks of potentially sensitive information via these providers. Threema even felt compelled to take a concrete stand on the topic of push notifications due to these revelations. They directly addressed their protection mechanisms in their app [6].

As FCM is based on Google's cloud infrastructure, the security implications of cloud access also apply to push notifications. Past leaks of such data may also include sensitive information used in push notifications [7].

C. RESEARCH GAP

Even though the importance of secure messages with FCM is high, to our knowledge, there is no overview of securing mechanisms "in the wild". Some apps directly advertise mechanisms like "end-to-end encryption" (e2ee), which implies that messages transmitted via FCM are encrypted. It is hard for end-users to determine how push notifications are protected. The most user-facing information regards the visibility of notifications on the lock screen and is often labeled with the keyword "privacy" [8]. Pleas to add more protection in push notification systems even go as far back as 2014 [9]. Google provided a library for this purpose but abandoned development shortly after publishing it to the general public [10].

With this work, we try to understand if real-world Android apps send sensitive information via push notifications and how app developers protect their push notifications.

D. RESEARCH QUESTIONS

To shed light on the identified research gap, we design and conduct various studies to answer the following research questions:

RQ1 *"How ubiquitous is the use of push notifications in Android apps?"* We are interested in understanding the extent to which push notifications are used in Android apps in practice. In particular, we want to know if Firebase Cloud Messaging (FCM) from Google has a dominant position among apps.

RQ2 *"Are there currently any security or data protection problems with push notifications in practice?"* Push notifications are crucial for users' security and privacy if they contain unprotected sensitive data. We are interested in gaining insights into whether apps consider the specifics of the communication channel for push notifications and adequately protect sensitive data from end-to-end.

[1]The corresponding implementation on Apple systems is not considered.
[2]https://protobuf.dev/

RQ3 *"Can we identify common practices for protecting push notification messages in Android apps?"* We want to find out whether we can identify common practices and evaluate their characteristics based on the protection measures found in practice. Based on the knowledge gained from this research question, we want to provide software developers with better support for the protection of push notifications.

E. METHOD SUMMARY

We performed a mixed-methods study to address the above research questions (cf. Figure 2). First, using experimental app prototypes, we identify features that allow us to detect the use of FCM-based push notifications through static analysis of APK files. Based on the derived features, we then statically analyzed 100,000 Android apps for the use of the push notification provider FCM, finding that more than half of them use it. Subsequently, we selected applicable apps from a subset of 400 apps for an in-depth exploratory manual analysis. We then used qualitative analysis techniques to gain insights into the protection of push notification content and to identify common practices.

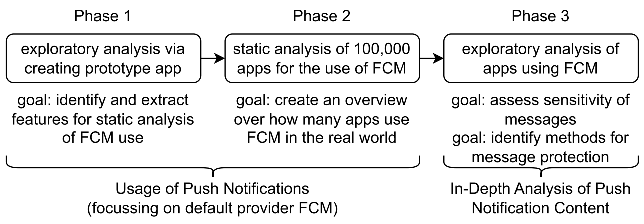

Phase 1
exploratory analysis via creating prototype app
goal: identify and extract features for static analysis of FCM use
Phase 2
static analysis of 100,000 apps for the use of FCM
goal: create an overview over how many apps use FCM in the real world
Phase 3
exploratory analysis of apps using FCM
goal: assess sensitivity of messages
goal: identify methods for message protection
Usage of Push Notifications (focussing on default provider FCM)
In-Depth Analysis of Push Notification Content

FIGURE 2. The analysis is divided into two large blocks using a mix of methods: phases 1 and 2 answer RQ1 and phase 3 focuses on RQ2 and RQ3.

F. CONTRIBUTIONS

Conducting the presented mix-methods study, we provide the following contributions:

1) *Large Scale Analysis.* We provide a large-scale overview of how many apps from the Google Play Store use Firebase Cloud Messaging (FCM) based on an open data set of 100,000 most-rated apps. This includes the methods "Data Message" and "Notification Message", which are handled differently by the operating system.
2) *In-Depth Investigation of Payloads.* An in-depth qualitative analysis giving insights into the real-world content and usage of notification payloads. This also includes insights about whether sensitive data is transferred in push notification messages.
3) *Common Practices to Protect Payloads.* We found that approaches to protect push notification messages can be categorized into "custom protocols", "out-of-band", and "encryption". However, even apps in the same category often differ in their implementation.
4) *Replication package.* We provide the data to replicate our static analysis results and also include our intermediate results for the qualitative in-depth analysis.

G. PAPER STRUCTURE

The remainder of this paper is organized as follows: Section II discusses previous research results regarding Android security, privacy in relation to push notifications. In section III we detail the static analysis of 100,000 Android apps and give an overview of the current usage of FCM in the wild. Section IV explains the in-depth manual analysis of 60 apps and shows the results indicating which protection schemes are used. In section V we discuss the limitations of our methods, discuss our work in section VI, and conclude our paper in section VII.

II. RELATED WORK

We discuss related work in four key areas: First, information leakage to third parties, meaning service providers and libraries, such as analytics and ads. Second, information leakage in the context of push notifications. Third, network security concerns possible eavesdropping methods to obtain sensitive data without direct access to the device or server components. Fourth, there are challenges in deploying protection schemes to protect against some of the previous key areas from the end users' and developers' perspectives.

A. LEAKING SENSITIVE PERSONAL INFORMATION TO THIRD-PARTIES

The realm of on-device privacy leakage is extensively researched: this includes sensitive data leaking to third parties by integrating analytics libraries [11], [12]; poor logging practices [13]; ad libraries [14], [15]; and payment methods [16], [17] into an app. Other apps on the same device can leak data by going through side-channels or issues in the permission system [18]. Push notifications allow apps to read other apps' notifications and leak information through the displayed text [19].

Leaks during data transport through cloud infrastructure happen when the data is not properly secured, either by the provider or the app developers. Issues like insufficient access control can lead to remote control of devices and sensitive data leaks [20]. The reasons for such issues are often misconfigured cloud back-ends or app servers, e.g. when credentials are found in apps that can be used for tasks that are not meant to be accessible [21], [22].

Our delta: Previous research deliberately excluded transit notification providers and did not analyze the payloads of notifications [2]. We look into the payload of push notifications instead without considering on-device data. We explicitly only investigate FCM (what they call a "Transit Notification Platform") and exclude library analysis of other notification providers.

We do not investigate on-device privacy concerns regarding end users (e.g., malicious apps, or sharing a physical device with others), as the examination is solely focused on the internal data processing of notification payloads. However, we are looking into the data shared with the push notification service provider without exploring the question of whether or not users perceive it as a problem to their privacy.

We also excluded the avenue of data leakage from the cloud providers' servers. However, it gives motivation for properly protecting notification payloads. Providers could combine metadata and sensitive data to create detailed profiles for apps and users. However, we excluded any analysis of countermeasures for mitigating metadata transmission and solely focused on analyzing the payload of push notifications.

B. USERS' ISSUES WITH PUSH NOTIFICATIONS

Users perceive the privacy of notifications as the visibility of notifications on the device itself e.g., on the lock screen or when giving the device to another person. They associate privacy with the settings for the visibility and content of these messages on the screen [8]. A high number of apps can be found that use aggressive notifications mainly for advertisement purposes potentially annoying the user so much that they remove the app [23]. Security issues come into play when notifications trick users into action by impersonating other apps e.g., for phishing [24].

Our delta: The perception of end users is irrelevant to analyzing payloads of notification messages, as protection methods are not visible to the end users, and they cannot control protection schemes inside an application. Additionally, this work is out of scope for the design and content considerations of displaying notifications on end users' devices.

C. NETWORK SECURITY IN ANDROID APPS

Tracking devices by recording clear-text client certificate authentication of connections was easy to achieve and had implications on the development of new TLS versions [25]. But even on encrypted connections, eavesdropping can enable user tracking by exploiting publicly visible notification triggers on social networks. Recording, when notifications are triggered, can correlate devices to users, circumventing the privacy protection of transport encryption [26]. Even though using the tor network to transport push notifications mitigates some of these risks, it incurs heavy costs even with improvements, e.g., network usage and battery drainage increase [27].

Our delta: As FCM uses TLS-secured connections, we consider the potential of leaking private data via eavesdropping low. It only concerns metadata such as timings or payload length, but the content is secure from wiretapping parties. We prioritized push notification service providers as they have clear-text access to the payload.

D. CHALLENGES OF DEPLOYING PROTECTION SCHEMES

There are tools available for app developers to check their used libraries for privacy issues [28]. Integrating SDKs from push service providers can introduce privacy issues, either by the app developer's integration of the SDK or internal issues inside the SDK itself. Nevertheless, 20 % of apps in the Google Play Store were affected by such issues [29].

Developers need the right tools and knowledge to integrate protection schemes like e2ee into their software. Such tools and libraries for developers need improvement [30]. Still, tools are already available for helping developers using cryptographic APIs that flag misuse and potential security vulnerabilities [31]. However, improving the developer experience can increase the positive influence of real-world applications of such tools. This requires creating a better understanding of developers and overcoming wrong assumptions by us researchers [32].

Developers need to consider end users, as the usability of e2ee implementations in instant messaging is essential. End users need to understand it to use it effectively [33]. Sometimes, a wrong perception of such encryption mechanisms can be created by developers who do not consider the mental models of their users [34]. Often users think that their messages are not protected well enough and feel vulnerable, even though the technical implementation protects them [35].

A negative example is email encryption, which has to be enabled on top of existing infrastructure, and the spread of encryption is hindered by end-user usability and unawareness of the issues [36]. From a developer-centered viewpoint, the same issues arose when SSL was introduced to replace clear-text transmission [37]. On Android, a similar issue is certificate pinning, which falls into the responsibility of the app developers [38].

If we want to introduce a protection scheme, we need to take a developer-centered view and take the burdens of app developers into account [39]. This is not limited to library design and includes usage patterns when adding tools into existing apps [40]. An ideal approach is similar to a default option and can be easily integrated into existing development practices of organizations [41].

Our delta: During the static analysis of apps and the in-depth analysis, the aspect of end-to-end encryption is touched upon. During the static analysis, we searched for potential encryption libraries; during the in-depth analysis, we saw some protection schemes that included encryption. However, we only searched for such schemes and did not evaluate specific encryption methods.

III. USAGE OF PUSH NOTIFICATIONS

In this section, we answer **RQ1** and determine the number of apps that use push notifications via FCM to gain initial insights into the extent to which push notifications via Android's default push service are widespread in practice. In addition, we are also interested in understanding what type of push notifications are commonly used. For this purpose, we developed a static analysis pipeline, which we describe in the following subsections.

A. METHODOLOGY

We developed and followed the methodology shown in Figure 3 to analyze the use of FCM push notifications in Android apps. First, we had to develop a method to determine if an app uses push notifications via Google's FCM service. In the absence of available techniques, we used various self-developed prototype apps to identify and extract features. Based on these, we developed static analysis tools to automatically analyze APKs and document whether they use FCM push notifications and if so, what type of notification they use.

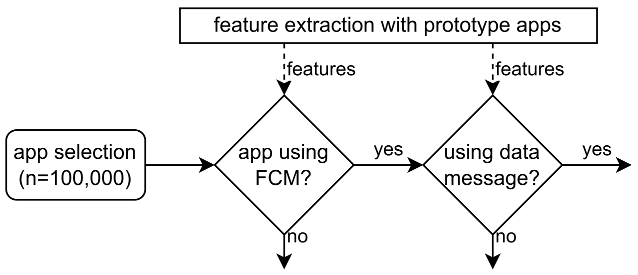

feature extraction with prototype apps
features
features
app selection (n=100,000)
app using FCM?
yes
using data message?
yes
no
no

FIGURE 3. The developed static analysis pipeline to determine the extent to which push notifications are implemented using Google's FCM service and whether they use Notification or Data Messages.

1) APP SELECTION

We used the AndroZoo dataset as the source for APK files, as it contains versioned APKs with unique identifiers and allows others to reproduce our work and results [42]. We selected 100,000 apps and considered it to be a reasonable number because only 13 apps have download numbers below 1,000 and the lowest rating count is 322. This indicates a saturation of the data set with apps that have at least some users who also left reviews as an indication of engagement with the app.

More precisely, we used the AndroZoo dataset of October 26, 2023, which contains 23,518,822 APKs. We reduced the number of APKs by selecting apps from the Play Store and choosing the newest version of each app. Additionally, we only consider apps that received an update in the last year (in relation to the most recent APK entry in the dataset). We then sort these apps in descending order according to the number of ratings (regardless of the actual rating value) and select the 100,000 apps with the most ratings.[3]

With this approach, we select apps that are not only frequently downloaded, but also rated by users. This minimizes the number of apps that are used seldom or not at all resulting in apps that have a tendency to be used more frequently and are important enough for users to rate them. The selection may disadvantage newer apps, as they generally still have few ratings due to their limited time in the Google Play Store.

[3]We provide a list of these apps as supplementary material.

Still, our selection ensures that the analyzed apps have a significant impact on users and provide valuable insights into the actual app usage of push notifications.

Note that the entries in the AndroZoo dataset only contain base-APKs, even for app bundles, and therefore no potential feature modules that could contain code for an app's functionality. However, we used hints in the manifest file contained in the base-APK, so determining FCM usage does not require feature modules. If we could find indications of FCM in the manifest, but no code for processing notification payloads, then either no push notifications are processed or the code for this is located in a feature module. In both cases, the corresponding APK is not considered for further analysis.

2) DETERMINE FCM USAGE

Figure 4 shows the process of inspecting the manifest file and package list of an APK to determine if it uses FCM push notifications or not.

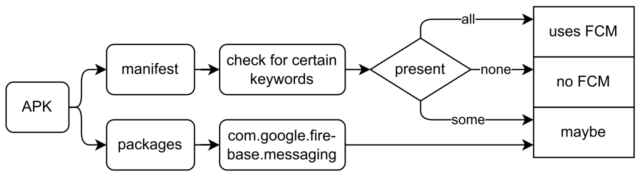


FIGURE 4. Process flow delineating the steps for detecting the use of FCM push notifications in an APK.

The manifest file included in the APK of each app contains entries for permissions and settings, including predefined directives for FCM. This means we only need to analyze the static metadata of an app's APK, so no resource-intensive code analysis is required, and the analysis of the manifest is not as error-prone as disassembling an APK.

To ascertain the required manifest entries for FCM, we referred to the developer documentation and crafted a sample app from scratch [43]. By systematically adding and removing FCM directives in the manifest file while verifying FCM functionality with the prototype app, we pinpointed the essential keywords for FCM. Ultimately, we identified these four essential strings to receive push notifications for both Data Messages and Notification Messages:

- `com.google.android.c2dm.intent.RECEIVE`
- `com.google.firebase.messaging.FirebaseMessagingService`
- `com.google.firebase.MESSAGING_EVENT`
- `com.google.firebase.iid.FirebaseInstanceIdReceiver`

If all four of these directives are present in the respective section of the manifest file, the app is technically equipped to receive push notifications via Google's FCM. We denote such apps as "uses FCM". If none of these strings are present, the app cannot receive any push notifications from FCM and we declare such apps as "no FCM". If only one, two, or three strings are present or the list of imported packages contains the name of the FCM library, we declare them as "maybe". This case occurs when developers either do not correctly add or remove FCM library code in their apps, so these apps contain remnants of the keywords, but cannot receive notifications.

3) CHECK FOR DATA MESSAGE

Next we want to distinguish between "Data Messages" and "Notification Messages". Figure 5 shows the analysis process of an APK that is categorized as "uses FCM".

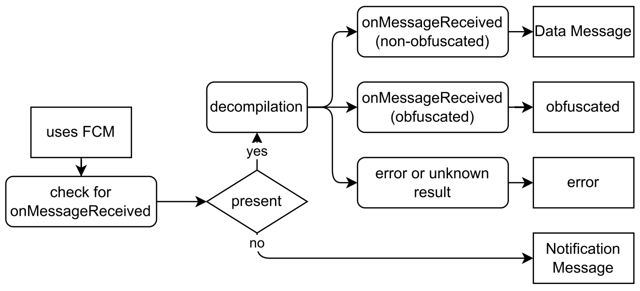


FIGURE 5. **To determine whether Data or Notification Messages are used, we check for the presence of the necessary onMessageReceived method inside the app. Then we determine potential obfuscation techniques by checking if the required method is visible as plain-text.**

To receive the data of a Data Message within an app, developers must implement a corresponding method by overwriting the *com.google.firebase.messaging.FirebaseMessagingService.onMessageReceived-(RemoteMessage)* method. They need to declare their implementation method in the manifest file, so we use it as the entry point for our analysis. If there is no receiver method declared in the manifest of the app, notifications are treated as "Notification Message" by the FCM library and the operating system and we consider them unprotected. If an implementation of the `onMessageReceived` method is available, we try disassembling it, resulting in the source code, obfuscated code, or an error.

As the final step requires access to disassembled code, only successfully disassembled plain-text code is relevant. We refrain from de-obfuscation, as the analysis aims to provide a general overview. Our analysis revealed, however, that the number of non-obfuscated code exceeded the number of obfuscated code which is in line with previous research [44].

B. RESULTS

The selection of 100,000 apps produced a set of apps ranging from all possible number of downloads used by the Play Store. The distribution of apps (cf. Figure 9 in the appendix) suggests that all are used by more than one user, as only 13 apps have sub-thousand download counts. The minimum number of ratings was 322, and the maximum was 178,886,403.

Note that all apps with Notification Messages are, by definition, unprotected, as no further processing of notifications takes place inside the app. However, apps with the label Data Message (with or without obfuscation) can also additionally receive Notification Messages.

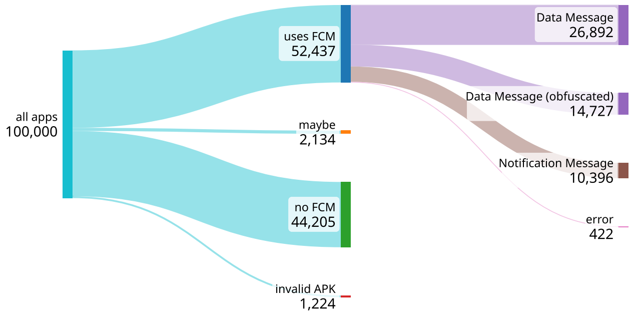


FIGURE 6. **The static analysis results show if apps are using push notifications and how their usage is distributed.**

The results show that more than half of all analyzed apps use FCM in some form (cf. Figure 6). Out of these apps ca. 80 % use Data Messages and some of them even obfuscate the code in some form. These apps are of interest for further explorations, as they can implement custom handling of push notification content and therefore have the ability to implement protection mechanisms.

All apps using FCM combined have a minimum number of downloads of 287 billion with a minimum of 1, a maximum of 10 billion, an average of 5.5 million, and a median of 100,000. Note that the Google Play Store only gives an estimate of the number of downloads: e.g. "10B+ Downloads" for Google Chrome. However, some are standard Android apps[4] and therefore have an artificially high number of downloads.

TABLE 1. **App categories sorted by their above-average usage of FCM. "All" denotes the percentage of apps in the specified category in the 100,000 apps set. "use FCM" is based on the overall "uses FCM" set, as seen in Figure 6. Note that all game sub-categories e.g., "strategy", "racing", or "action", are merged into one.**

Category	All	use FCM	diff
Finance	4,89 %	7,66 %	2,78 %
Shopping	2,24 %	3,93 %	1,69 %
Business	2,16 %	3,36 %	1,20 %
Lifestyle	3,10 %	4,14 %	1,04 %
News & Magazines	1,42 %	2,45 %	1,03 %
Social	1,71 %	2,62 %	0,91 %
Travel & Local	1,94 %	2,73 %	0,79 %
Sports	1,44 %	2,20 %	0,76 %
Food & Drink	1,02 %	1,71 %	0,70 %
Health & Fitness	2,68 %	3,22 %	0,53 %
Communication	1,66 %	2,12 %	0,46 %
…			
Music & Audio	3,16 %	2,74 %	-0,41 %
Photography	2,03 %	1,52 %	-0,51 %
Personalization	3,07 %	2,26 %	-0,82 %
Tools	7,67 %	5,97 %	-1,69 %
Games	29,02 %	19,28 %	-9,74 %

Table 1 shows which categories use FCM disproportionately more or less relative to their distribution among all apps.

[4]like built-in mail or browser apps.

For example, shopping apps make up 2,24 % of all apps, but out of all apps that use FCM 3,93 % are shopping apps. The difference between both values shows that apps from a category tend to use more FCM than apps from other categories.

Categories are self-selected by app developers: the same types of apps might be found in different categories, depending on the assessment of the developers. The most notable outliers are games that not only use less FCM than their proportion of all apps suggests but use disproportionately more Data Messages instead of Notification Messages. On the other hand, finance apps use FCM more often than their share suggests.

TABLE 2. FCM usage per download category as displayed in the Play Store. Note that the number of downloads is not linear and that some download categories only contain few apps. The age is based on the first time an app was seen by the crawlers of the AndroZoo data set which has a lowest value of 2020-05 [42].

Downloads	Number of Apps	use FCM	avg. Age
1+	2	50 %	2020-05
100+	5	00 %	2023-01
500+	6	67 %	2022-08
1K+	218	64 %	2022-08
5K+	549	63 %	2022-03
10K+	7059	61 %	2021-07
50K+	9107	57 %	2021-05
100K+	31022	53 %	2021-04
500K+	12919	51 %	2021-02
1M+	23033	50 %	2021-01
5M+	6038	51 %	2020-12
10M+	6918	52 %	2020-11
50M+	1038	57 %	2020-08
100M+	697	64 %	2020-08
500M+	77	71 %	2020-07
1B+	64	66 %	2020-06
5B+	17	41 %	2020-06
10B+	7	57 %	2020-07

Table 2 shows that apps with downloads in the range of 500,000 up to 10 million exhibit the lowest percentage of apps with FCM features found. The extremes of the range with few or very many downloads seem to have more apps with FCM usage within their categories, forming a kind of bathtub shape. The average age of the apps decreases with the number of downloads. However, the usage of FCM does not seem to correlate with age (cf. table 7 Appendix B), as the Pearson value of −0,29 might even suggest a decrease in FCM usage over time [45].

FCM is widely used, regardless of whether an app has been downloaded frequently or not and age hardly seems to have any influence on this, if anything there may be a slight decline. Note that our data set is skewed towards older apps, which results in lower sample counts for newer apps (cf. table 7 Appendix B). We did not analyze if apps generally add features over their lifetime and cannot say whether FCM might be a feature that is added to more apps as they get older.

The results show that FCM can be found in every second app and is, therefore, a worthwhile research target. Its share of more than 50 % among all apps (apps with and without push notifications) shows that it can be considered the leader for push notifications on Android. Ca. 80 % of apps using FCM seem to use Data Messages, which allow apps to process their notification payloads and potentially employ protection schemes. Unfortunately, ca. 30 % of apps using FCM seem to obfuscate their byte code and, therefore, were not further analyzed.

RQ1: How ubiquitous is the use of push notifications in Android apps?

- More than 50 % of the analyzed 100,000 apps utilize push notifications via Google's FCM service.
- Approximately 80 % of the analyzed apps utilizing FCM employ Data Messages, enabling them to process the notification payload. The remaining 20 % use clear-text Notification Messages.
- Some categories use disproportionately more push notifications, notably "finance" apps.

IV. IN-DEPTH ANALYSIS OF PUSH NOTIFICATION CONTENT

Our in-depth analysis aims to answer **RQ2** and **RQ3**: find common behavior of apps using FCM notifications and whether they protect their payloads. We test apps by triggering notifications, capturing their content, analyzing it, and categorizing them based on our observations of their network traffic and checking for sensitive information that might be leaked.

A. METHODOLOGY

The methods used for analyzing apps with a manual analysis are separated into multiple parts: app selection, a traffic capture setup, and the per-app analysis flow, including triggering notifications and analyzing captures. During the analysis, we took the position of a push notification service provider. By intercepting the network traffic between a smartphone and FCM servers, we can analyze the content that the service provider is also able to see.

The process of triggering app notifications is manual and cannot be automated. It is unique to every app and depends on the context of usage. Most apps require an account to be usable, which also cannot be automated. Accounts must be created manually, often requiring additional verification steps like mail or SMS verification. Therefore, the methods for capturing evidence are entirely manual and have influenced our decisions regarding the selection of apps.

1) SELECT APPS

The manual process of analyzing an app is time-consuming so we had to make compromises selecting apps. In practice, we are limited to apps that do not require more information than an e-mail or phone number. In particular, we cannot easily create accounts in country-locked or financial apps, which often require specific information or additional accounts from other services. When we started with six finance apps using personal accounts from our research group, we found that it was exceptionally hard to trigger notifications. Two did show notifications on money transfers and one used certificate pinning, preventing us from gaining insights into the network traffic of the app. We concluded that the effort is too high to be able to analyze a relevant number of finance apps.

Communication Apps: To select apps for the manual analysis, we combined two lists derived from the results of the static analysis filtered for apps using FCM. The first is a list of the 200 most rated apps from the category "communication" because we suspect instant messaging apps are in this category. A first quick check of the Play Store showed several popular chat apps (WhatsApp, Facebook Messenger, Telegram, and Signal) in this category. We also assume that apps from this category send more FCM notifications overall, e.g. each time a message is sent. This assumption would also mean that a high percentage of notifications going through FCM services originate from messaging apps, making this category of apps more vulnerable to a threat from FCM. We cannot confirm our assumption about messaging apps using FCM more frequently than other apps as it would require numbers from internal FCM services, which we cannot obtain.

We wanted to increase the number of messaging apps as these typically provide easy account registration and only require mail or phone numbers. Furthermore, they have similar workflows for analyzing app notifications: connect two accounts (which might trigger a notification on "friend request") and then send a self-selected string to another account, which triggers a notification.

Additionally, these apps often have similarly sensitive notification content. They show usernames and user-defined strings, meaning a high probability of private information.

Popular Apps: The second list contains the 200 most rated apps from the Play Store, which are not in the categories "communication" or "games". We excluded games for several reasons: we suspect that triggering notifications is more complex, they probably do not contain sensitive data in their notifications, they make up nearly 30 % of apps in the data set used in the static analysis, they could flood the app selection, and their use of notifications is disproportionally low. All these factors make this category an outlier that should be handled separately.

Combination: We combined both lists and added 11 apps (cf. table 5), because some researchers could provide private accounts for these apps. Afterward, we filtered for the presence of messaging functionality. We also checked if these apps are localized to specific countries or companies, meaning no account can be created without additional resources. This check was done by manually visiting each app's Play Store page and looking for hints of such functionality in images of the interface or the description. If in doubt, we classified an app as "yes, contains messaging" rather than not, as we then analyzed it further in the following steps.

2) CAPTURE NETWORK TRAFFIC

To capture network traffic, which includes the TLS-protected long-running TCP connection to FCM, we devised a simple setup with two smartphones connected via a test bed to the internet. We used Google Pixel 7 devices instead of an emulator, which allows the usage of real SIM cards in case the registration requires a phone number.

We used an entity-in-the-middle proxy (mitm-proxy; cf. Figure 10 in the appendix) to record all network traffic emitted from a device [46]. The operating system and apps must use a self-signed certificate of the proxy to decipher TLS traffic. Therefore, we rooted one of the Android devices to write this certificate into the system and user stores. The Play Services and most apps use this store, so we can see the content of the FCM connection. If an app uses certificate pinning, the traffic cannot be deciphered, and the mitm-proxy drops such connections. Apps that use certificate pinning were not altered and we did not try to remove the pinning. We kept the smartphones on Android 13 as it was the latest version during the development of the test bed, and Android 14 made it harder to alter the system certificate store.

During our study, mitm-proxy could not capture all UDP traffic, so QUIC and, subsequently, HTTP3 traffic could not be analyzed. As FCM only uses a TCP connection at the time of writing [47], we do not consider this shortcoming to be of high relevance. It does, however, prevent us from analyzing all HTTP traffic and means some out-of-band traffic could not be thoroughly analyzed.

3) RECORD SCREENS

In addition to the network capture, we also record the screens of our testing devices. The recordings are started right before the network capture and can be used to correlate the timings between both recordings. This enables us to identify the timestamp of on-screen notifications and the exact text displayed to the user.[5]

4) TRIGGER PUSH NOTIFICATIONS

Using two smartphones allows us to install apps and create accounts on both devices to trigger notifications by interacting with two accounts from the same service. Figure 7 shows the workflow for analyzing one app and was used by all researchers to create captures.

The general workflow consists of four phases:

1) app installation
2) account registration
3) recording and triggering of the notification
4) account and app deletion

[5]Note that the screen recordings show UTC+1, and the traffic captures use UTC due to a misconfiguration.

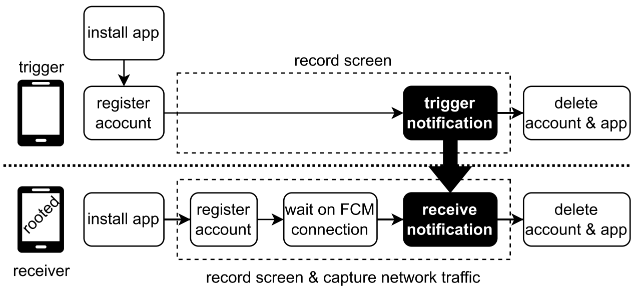


FIGURE 7. Workflow (from left to right) for triggering a push notification on one device (trigger) and receiving it on another device (receiver). The triggering and receiving of a notification take place while a screen recording and network capture are running.

The app installation is based on three sources. First, we try to execute the APK from the AndroZoo data set directly, as this is the same APK file used during the static analysis. Some apps cannot start when only the base-APK is installed, they either crash or hang during startup. If this occurs, the same app will be installed via the official Play Store app. Sometimes, the Play Store refuses to install an app due to country- or age restrictions. Then we use the Aurora Store instead, which proxies the Play Store and uses the same catalog. Only three apps failed on all attempts at installing them.

To test whether account registration is possible, we create the first account on the non-rooted device with unencumbered access to the internet. If this fails, we exit the analysis early and do not create captures. It also acts as a baseline if network issues arise on the rooted device, as they can be quickly ruled out as originating from the proxy setup.

5) ANALYZE EVIDENCE

We check the network capture for any strings used in the test run to categorize the data sent via push notifications. These strings are excerpts of the names used for testing accounts, e-mails, phone numbers, and the test string "testmessage" for self-selected texts. We then classify the protection of the payload and whether the notification contains sensitive data.

Network Capture with Notification: We use the screen recordings to get the timestamps of the creation of a notification trigger (e.g., sending a text message), the timestamp when the notification is shown, and the content of the notification. This allows us to select an appropriate time frame from the network capture and extract all traffic related to the notification. Traffic might include messages on the FCM connection on port 5228 or non-FCM messages from different TCP or HTTP connections but does not include UDP traffic apart from DNS queries.

Table 3 shows the process for analyzing the payload and categorizing its protection. It reveals how we used an inductive method to categorize the results: every time we found a new case (row or column) during our analysis, we added it to the matrix and considered what this meant for the existing categories.

FCM Contains Plain-Text String: Based on the recorded notification on the receiving device, we search in FCM messages for the same strings. As FCM uses Protocol Buffers to encode messages, such strings can be found as clear-text after the TLS connection is decrypted. If necessary, we analyze them manually and try to decode their data using decompression or simple decoding schemes such as base64. If we find strings, we note what data they hold (username, text, …). Payloads that contain partial clear-text data are categorized as "clear-text" (*), but only if this data is deemed sensitive (see below).

FCM Contains Unidentifiable Data: If there are FCM messages without clear-text, we extract the FCM messages we deem essential for the notification: Their identifier contains the app I-D, and the time of the recording corresponds to the screen recording. Suppose we find meaningful content, i.e., binary blobs big enough to at least hold the notification text in bytes, and no further messages on other connections were made. We categorize those as using "encryption" (§).

(HTTP) Request in Conjunction with FCM Message: When a (HTTP) request seems to be triggered by an FCM message, we check it for the strings in the notification. If true, we assume an "out-of-band" (†) scheme and categorize it as such.

If a message is received on a non-FCM connection and no FCM message is recorded, we categorize it as a "custom protocol" (‡) regardless of the actual content of the non-FCM message.

A particular case is an FCM message that does not trigger a notification on the device. This indicates "certificate pinning" (¶) when the FCM message triggers an out-of-band connection that fails due to the proxy. If notifications work without the proxy, this is an additional indication.

No FCM or Other Messages: If we do not see any messages in the network capture, but a notification is shown on the

TABLE 3. After extracting the network traffic from a captured notification, we analyze both FCM and non-FCM traffic separately. Depending on the combination of the results of both data streams, we categorize apps differently to see what protection schemes are used (cf. RQ3). Empty cells are conditions that cannot result in a notification.

		FCM			
		plain-text string	unidentifiable data	empty	nothing captured
Non-FCM	plain-text string	*	†	†	‡
	unidentifiable data	*	†	†	‡
	aborted connection	*	¶	¶	
	nothing captured	*	§		

* = clear-text FCM push notification (30)
† = FCM triggered out-of-band notification (3)
¶ = out-of-band with certificate pinning (3)
‡ = custom protocol triggered notification (17)
§ = encrypted FCM notification (7)

screen, we suspect the notification was triggered locally. It might also indicate an error during recording. Therefore, any recordings in this case are "not categorized".

What is sensitive information? We define sensitive information as data that might relate to a natural person and per-user-specific information, like user-generated or personalized content. This includes:

- usernames, names, and other name-like descriptions, including user IDs.
- user-generated content like user-defined text messages or similar.
- personalized information like transaction information or non-general descriptions, including security codes.

We consider other information that seems like general strings provided to multiple users, e.g., advertisements or welcome messages without usernames, insensitive. If we recorded multiple notifications, we combined their sensitive data.

B. RESULTS

Out of the 411 apps considered for qualitative analysis, we selected 109 apps that might contain some form of messaging feature and added manually selected apps. 8 apps offered messaging features on the Play Store page, but a further look after installation turned them into "does not contain messaging feature". In total, we manually inspected 113 apps, but could only analyze 60 of them for their push notification usage (cf. table 4). All other apps either could not be installed (3; 2.7 %), no user account could be created (18; 15.9 %), or we could not trigger a notification (32; 28.3 %).[6]

TABLE 4. The categories of payload protection after we did an in-depth analysis of 113 apps and recorded 60 apps that created notification data.

Category	Apps	
clear-text	30	50.0 %
custom protocol	17	28.3 %
(end-to-end) encryption	7	11.7 %
out-of-band without certificate pinning	3	5.0 %
out-of-band with certificate pinning	3	5.0 %

Half of the 60 apps for which we could trigger and capture a notification do not protect their messages at all. Two apps used simple base64 and ProtoBuf encoding inside the FCM message payload but no further protection schemes.

1) CUSTOM PROTOCOLS

Most of the apps using some form of protection for their messages implement a custom protocol. This suggests that the motivating factor is not the protection of the payload itself but the decision to use custom protocols as the transport mechanism throughout the app. Custom protocols vary widely in their implementations and are app-specific by design. We saw various protocols from text-based to binary and using well-known encoding schemes like JSON or ProtoBuf and unknown binary formats. However, at least four apps use XMPP-derived protocols, a common protocol for messaging applications.

A special case is FCM as a fallback to this custom protocol: we analyzed WhatsApp and found that FCM messages are sent when no XMPP connection can be made between the app and the application server. Then the FCM message merely wakes up the app, which then connects via its custom protocol. If not, a notification with the text "You may have new messages" indicates this state to the user. Telegram and Rakuten Viber Messenger showed the same behavior with their FCM messages.

One app used a custom protocol that sends its notification payload in clear-text over an unprotected and unsecured TCP connection. This shows that implementing a custom protocol does not automatically increase payload protection or can even be worse than using FCM. A custom protocol puts an additional burden on the app developer and increases the risk of implementation issues.

Similarly, another app used a custom protocol to request the notification information out-of-band, triggered by a mixed Notification and Data Message. However, the Notification Message contained the message sender's phone number in clear-text, while the custom protocol loaded the username and text message. The final notification only displayed the username and text message, no phone number was displayed. Therefore, it was unnecessarily transmitted and unprotected.

2) OUT-OF-BAND REQUESTS AND END-TO-END-ENCRYPTION

Our analysis's categorization of out-of-band requests is somewhat blurry as a custom protocol for doing a follow-up request after an FCM message is considered out-of-band. Consider the two cases: (a) a notification is shown on the screen, no FCM traffic was captured, and traffic on a custom protocol was captured; and (b) a notification is shown on the screen, FCM traffic was captured, and traffic on a custom protocol was captured. Only if the custom protocol seems to trigger a notification independently of FCM usage (a) do we categorize it as "custom protocol", otherwise (b) we consider it to be an out-of-band request. This means some apps using a custom protocol fall back to an out-of-band scheme where the FCM message wakes up the app and triggers a new connection. Custom protocol connections might be dropped when an app is forced into the background by the operating system.

Only around 20 % of the apps protect their payloads via encryption or out-of-band requests. Some notable examples are Facebook Messenger, which seems to use encryption for its FCM payload, and Signal, which uses out-of-band requests combined with nearly empty FCM payloads.

We consider encryption and out-of-band as common practice categories, as the apps' behaviors within each group

[6]The complete list of manually analyzed apps and if a notification could be triggered is available as supplementary material.

are similar. This contrasts with apps in the custom protocol group, as each protocol might behave differently. We could only identify a common practice for messaging-focused apps: if they use a custom protocol, they often use XMPP or similar. Additionally, if apps use a custom protocol, they need to keep a connection open, which the operating system might suppress to conserve battery. In such cases, apps seem to fall back to FCM to trigger a reconnect of the custom protocol.

One notable case is Facebook Messenger: our static analysis classified it as using Notification Messages, but our in-depth testing took place on a more recent version from the Play Store. Between these two analysis, the Facebook Messenger introduced end-to-end-encryption to its messages, so the results diverge.[7]

RQ3: Can we identify common practices for protecting push notification messages in Android apps?

- If apps protect their payloads, they tend to use custom protocols instead of only relying on FCM.
- Implementing such custom protocols brings other issues like unprotected TCP connections or relying on FCM to wake up an app.
- Only 20 % of apps protect their payloads with encryption or out-of-band when relying on FCM to trigger a notification.
- Encryption implementations differ from each other and do not show a common message format.

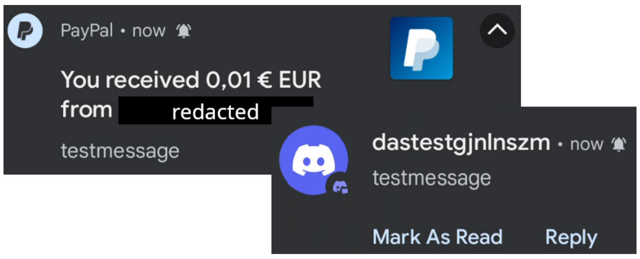
3) IMPACT

Our results (cf. table 6 appendix) show that some of the apps using clear-text have download numbers above one billion, Instagram even has more than 5 billion. Only two apps have numbers below 10 million. They accumulate over 12.8B downloads in the Play Store. The impact of such apps is high, not only because they transmit sensitive data like usernames and text messages but also because there are potentially a huge number of affected users. Most apps contain chat or email messages i.e., user-defined text. Figure 8 shows a practical example of such apps: their notifications contain sensitive information and spill it to FCM.

4) APPS WITH NOTIFICATION MESSAGES

Our in-depth analysis yielded only two apps that might use Notification Messages. Therefore, we created another list of 200 apps by filtering their results from the static analysis and only including apps with Notification Messages, excluding games. We then checked their Play Store pages for indications of messenger functionality or indications that they might handle sensitive data.

We saw 7 apps (without Facebook apps, because of their false reporting as stated above) that might include messaging


PayPal • now

You received 0,01 € EUR from redacted

testmessage

dastestgjnlnszm • now

testmessage

FIGURE 8. Two example notifications from apps that use clear-text payloads as seen on a device. This means that the content of the notification is also visible to the push notification provider.

features or are messaging apps and we suspect that they might handle usernames and text messages as sensitive data. We also saw 19 apps that seem to handle financial information, 3 apps that might handle health data and 4 apps that might handle some other potentially sensitive topic. However, it remains unclear if this information is sent via notifications. However, we did not analyze financial apps from this list for the same reasons as the previous in-depth analysis.

We manually analyzed 7 apps with messaging features. We could only create an account and trigger notifications for one app. It contains a messaging function and sends username and chat messages in clear-text. Our results of Notification Message apps, therefore, remain inconclusive.

5) SENSITIVE PAYLOAD

All 60 apps show some form of sensitive data. There is a bias in our testing process (cf. section IV-A4), as we tried to trigger notifications by an action that creates personalized text deliberately. In contrast, other sources of notifications might use public information, such as news or advertisements. However, we could not trigger such sources in the manually analyzed apps.

Apart from two apps, all included some form of name or username. 49 (81.7 %) also included a user-defined text or subject and potentially displayed more sensitive data, as opposed to a simple username. Additionally, we analyzed six financial apps (N26, PayPal, and four apps of German banks: comdirect, Sparkasse, VR-Bank, Postbank), which contain highly sensitive information: real names and amounts of money transferred. N26 used encryption and certificate pinning to protect the notification's payload. Still, it did not notify when a proxy was used, potentially indicating out-of-band usage on top of encryption. PayPal, on the other hand, did not protect its payload in any way, and we subsequently categorized it as clear-text.[8] Its payload contained the name of the sender and the exact amount of money transferred (cf. Figure 8). The other four apps we analyzed did not trigger a notification on transfer, even though our testing method was the same as with N26.

[7] https://about.fb.com/news/2023/12/default-end-to-end-encryption-on-messenger/

[8] We responsively disclosed our findings to PayPal.

RQ2: Are there any security or data protection problems with push notifications in practice?

- More than half of analyzed apps do not use any form of protection.
- All apps we manually analyzed included sensitive data, ranging from usernames and communication messages to financial transfer information.
- This allows the push notification service provider to combine data about users, and leaks at these providers theoretically put sensitive information at risk.

V. LIMITATIONS

Both studies have some limitations that should be considered when discussing their results. The app selection was made with some trade-offs, as it influences the feasibility of some analyses.

A. USAGE OF PUSH NOTIFICATIONS

The static analysis has limitations concerning the app selection and the specifics of our app analysis. Our approach was deliberately kept simple, as the only goal was to get a rough overview of the subject.

1) APP SELECTION

The selection of apps is biased towards apps that are used by many users. Selecting apps randomly would give a more representative example of the overall state, but fails to take the impact to users into account.

As the number of downloads is only available as a rough estimate and using it would favor pre-installed apps, we chose to use the number of ratings of an app instead. This filters apps that generate user feedback, whether good or bad, and we think it sufficiently represents importance to the users. Although the selection based on ratings might penalize newcomer apps (i.e. apps that had high growth in the last year of the data set), the high number of analyzed apps also includes apps with low download numbers, indicating that we saturated the data set.

2) STATIC ANALYSIS

Although the strings for FCM usage are systematically derived from a real prototype app, they only confirm the theoretical usage of FCM per app. If an app employs a library for push notifications, it remains unclear if all detected apps use FCM in a meaningful way.

A data flow analysis could have provided more insights into payload protection schemes by statically analyzing the apps. However, it would be difficult to identify the specific usage patterns that apps employ to protect their payloads, as we did not have any prior knowledge about specific implementations. To mitigate this limitation, we instead used a qualitative approach via the in-depth analysis of push notifications.

The AndroZoo data set does not contain metadata about categories from the Play Store. We crawled this data ourselves for all 100,000 apps, but 8,669 apps could not be crawled successfully. Their Play Store pages were removed between the AndroZoo data set timestamp and our crawling date.

B. IN-DEPTH ANALYSIS OF PUSH NOTIFICATION CONTENT

We selected apps based on our ability to test them for push notifications, which results in a disproportionally high number of apps with messaging features. We assume that such messaging features might include sensitive data, as their content is user-defined. However, we still based our methods on the research questions we wanted to answer.

1) APP SELECTION

We included apps that do not focus solely on messaging, but most apps containing such features are based on social interaction between multiple users with similar interaction schemes. This excludes apps that might be interesting from the perspective of real-world users, such as shopping and finance apps. Our lists are also biased towards apps rated a lot in the Play Store, which further reduces representativeness but increases impact.

2) GATHERING APP DATA

The screen recordings are meant to show what the researchers saw during interaction and notifications, but a few apps use an Android feature to hide themselves from such recordings. It hindered us from getting exact timestamps of notifications so that we could only guess which messages in the network capture correspond to which notification.

The entity-in-the-middle approach for decoding encrypted network traffic limits the network traffic captures. As this requires self-signed certificates, we depend on apps to trust our certificates. We can only reject traffic from apps that use certificate pinning or their certificate stores, as we did not modify any apps. Apps combining out-of-band techniques with pinning cannot be adequately analyzed and we can only guess that apps use this combination based on incoming FCM messages that do not trigger a visible notification.

Apps that primarily use a custom solution to trigger notifications might use FCM as a fallback option, which potentially exposes sensitive data only in specific network environments. We only investigated such cases for the most popular apps using custom protocols, so we might have missed some edge cases.

3) TRIGGER PUSH NOTIFICATIONS

We trigger push notifications by using two accounts in the same app on different devices. This limits our testing to apps that allow account creation via Google account, mail, or phone number. We excluded apps that do not allow two accounts to be linked or send messages/transactions between accounts. During analysis, some apps do not show any notifications even after several attempts, so it is at the researcher's discretion to cancel the analysis of an app when notifications can be triggered. Therefore, we might exclude apps that exhibited technical problems during our testing or contained bugs preventing us from using them correctly.

4) ANALYSIS

Our analysis is based on previously acquired artifacts, limiting us to this data. We cannot fully reproduce issues regarding certificate pinning, as our proxy did not record failed TCP connections. Therefore, we can only guess that an app might use out-of-band requests with certificate pinning when we do not see any notification but an FCM message.

The analysis of the recorded artifacts follows specific steps, but some are executed at the researcher's discretion. This includes analyzing payloads not transmitted in plain text in notifications when we had to reconstruct the content manually. If we could not decode the data, we might erroneously consider it encrypted. We also did not evaluate any encryption schemes or test them for vulnerabilities.

VI. DISCUSSION

We discuss our ethical considerations and findings related to our research questions and put them into context with related work.

A. ETHICS

Our studies have no human subjects, and apart from researchers, no person or their data was put at risk. Our first study was based on an open-source data set, so we did not have to crawl the Play Store and create traffic. We only crawled the Play Store for metadata and limited our requests to one per second to limit the drain on resources.

In the second study, we only created user accounts that do not require personal information to protect the researchers and the services from false data. We used randomly generated strings for usernames to reduce the potential of naming conflicts with real users. Some services block identifiers for future use, so this reduces the negative impact of such cases. We also deleted all user accounts as soon as possible after analysis to reduce the long-term effect of abandoned accounts.

If we would have found security vulnerabilities during analysis, we would have disclosed them using responsible disclosure. We notified financial app providers if we found clear-text notifications so that they could react if this behavior was not intended.

B. RESEARCH QUESTIONS

We could answer all of our research questions and the results allow us to contextualize our questions.

FCM usage: A lot of apps contain code to handle their push notifications by using FCM. This comes with no surprise, as this is Google's built-in default for providing push messages to apps. As a system component, it enjoys certain advantages over in-app solutions: it can wake up apps and its TCP connection is long-lived without aggressive battery-conserving methods interrupting connectivity. Its prevalence also shows that push notifications are ubiquitous: it is very likely to have apps that use push notifications installed, increasing the potential that some of them contain sensitive information.

We cannot say precisely how many developers of these apps directly use the FCM SDK or an abstraction on top of the FCM SDK, such as additional libraries or services. From a developer-centered perspective, it would be interesting to know whether or not developers are aware of the missing payload protection in their notifications. Most apps use Data Messages, which enables protection schemes to be integrated, but it remains unclear if the services and libraries used by developers expose such features to developers.

Finance apps are an interesting case, as apps from this category use proportionally more notifications than other categories and we suspect a high amount of sensitive data to be present. We analyzed six apps and found one app with issues, indicating that there can be problems even for widely used apps. This category warrants an in-depth look, however, it is also exceptionally difficult to test such apps at scale.

Data protection problems: More than half of the apps we analyzed sent clear-text push notifications, which leaked potentially sensitive information to the FCM provider. We conclude that there are problems with security and privacy regarding push notifications in practice, affecting a large number of users.

We show that we need to further research why apps do not protect their notifications, as we think that the push notification provider has no technical reason the be able to read notification content. If the push notification provider has access to unprotected payloads, it could even try to alter its content without the app knowing (maliciously or not). This might be high-effort exploitation requiring deep access to the push notification handling process but it could result in subtle and hard-to-find issues. Messaging apps that use end-to-end encryption for their messages protect them from the push notification provider, and others should implement protection too.

Problems with very sensitive information: A notable case of a popular app that transfers highly sensitive information (PayPal) shows that even highly proficient developers do not seem to be aware of these potential issues. But also highlights the impact of our findings: even very sensitive information is not safe from this kind of exposition to the push notification provider.

Issues when protecting payloads: Three notable examples show that some developers have issues implementing protection for their push notifications. One custom protocol uses TCP without TLS, and another app uses FCM to trigger out-of-band requests on a custom protocol, but the trigger notification contains a phone number. Another app uses end-to-end encryption, but the username is still accessible as clear-text.

The FCM connection is not protected by certificate pinning. When an app uses a custom protocol with pinning and uses FCM as a fallback with an encrypted payload, it circumvents the pinning when the fallback is used.

Common practices of messaging apps: If messaging apps use a custom protocol, they favor XMPP or something similar. They use FCM as a fallback, suggesting that the underlying application architecture uses XMPP. The protocol is standardized, multiple implementations are available, and it can be used by a mobile client and other clients. However, apps that use custom protocols use multiple protocols, and the app vendor is responsible for their implementation. They include server and client components and require additional implementation effort. Therefore, some apps had implementation issues, indicating that more standardized solutions might be preferable.

C. COMPARISON TO MESSAGING SECURITY

From a technical perspective, push notification protection can be achieved by using similar methods to those used by e2e messaging apps. Apps using e2ee inherently provide encryption for push notification messages, while others must add the exact mechanism between the app server and client devices.

Modern protocols like MLS introduce efficient e2ee even for big groups of members [48]. One benefit of using FCM's push approach is that it retains the ability to send payloads to many different clients. Such broadcasting behavior is problematic to protect by other methods, as out-of-band requests risk denial of service when many clients try to contact an app server simultaneously. Custom protocols do not provide a fan-out, as the app server needs to handle many concurrent connections.

A real-world example of using message encryption at scale is Web Push. Recent research on this technique did not show issues with payloads themselves, only web-specific issues regarding the usage of its APIs and cross-site request forgery [49]. This is unsurprising, as the corresponding RFC states that "messages sent using this protocol can be secured against inspection, modification, and forgery by a push service" [50]. The FCM platform could benefit from introducing a similar scheme as a past attempt by Google demonstrated [10].

Although the metadata was not part of our research, we can consider two cases concerning payload protection: using a custom protocol and encryption. When using a custom protocol to notify a client, FCM is left out, and no metadata is shared. However, if such an app falls back to FCM messages, it still leaks metadata. Using encryption to protect the payload does not improve any disclosure of metadata to the push notification provider.

D. RECOMMENDATIONS

We think that to introduce protection to apps that currently use none at all, it should be as close to a drop-in solution as possible.

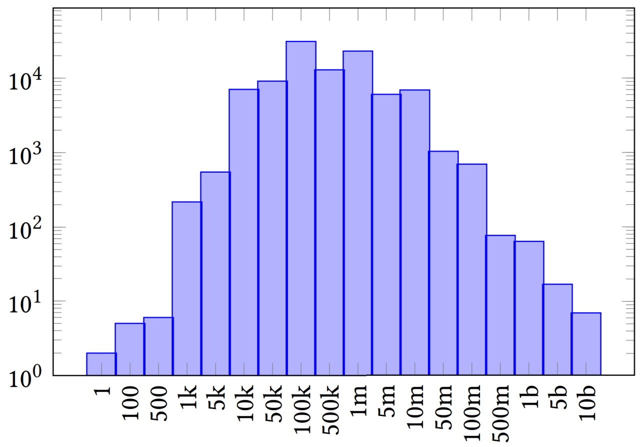


FIGURE 9. Distribution of apps in their minimum number of downloads categories in our static analysis data set. The minimum number of downloads as reported by the public Play Store page for each app.

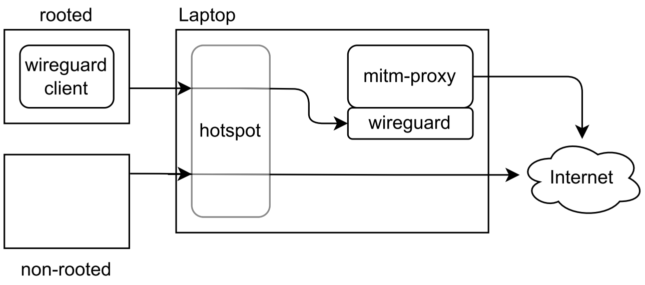


FIGURE 10. The networking capture setup with two smartphones as boxes on the left. The traffic of the rooted smartphone is routed through a Wireguard VPN to the mitm-proxy which captures all traffic including decryption of TLS.

For app developers: Using custom protocols can enhance the protection by reducing the impact of metadata leakage. However, similar to out-of-band requests, implementing such protection schemes requires more changes to an app and its server-side components than adding opaque encryption. Opaque encryption in the form of Web Push protocols can be retrofitted but leaves metadata unprotected.

For push notification platform providers: We suspect that not all app developers use FCM directly but through another service orchestrating push notifications. Ideally, such syndication services could opaquely implement encryption for app developers. Especially if Web Push is part of the syndication, it is possible to implement protected one-to-one notifications like Web Push was integrated.

For researchers: When considering how to improve the situation, we need to consider developers and their requirements. Ideally, a solution to a problem would be a drop-in replacement with no or very small changes to existing workflows. Adding protection is connected to the syndication of push notification techniques employed by different platforms, as most developers would probably prefer solutions that unify the handling of notifications on all platforms. In addition, we did not conduct a study on the

TABLE 5. 11 apps that were selected manually, as researchers from our group could provide private accounts for them.

Package Name	Name	Download
chat.delta	Delta Chat	100K+
de.tutao.tutanota	Private secure email Tutanota	500K+
network.loki.messenger	Session — Private Messenger	1M+
com.ebay.kleinanzeigen	Kleinanzeigen - without eBay	50M+
com.zhiliaoapp.musically	TikTok: Videos, Lives & Musik	1B+
de.number26.android	N26 — Love your bank	5M+
com.paypal.android.p2pmobile	PayPal - Send, Shop, Manage	100M+
de.comdirect.app	comdirect	1M+
com.starfinanz.smob.android.sfinanzstatus	Sparkasse Ihre mobile Filiale	10M+
de.fiduciagad.banking.vr	VR Banking - einfach sicher	1M+
de.postbank.banking	Postbank	500K+

awareness of developers about issues regarding their push notifications. We suspect that raising it would help with distributing solutions.

VII. CONCLUSION

In this work, we performed a static analysis of 100,000 Android apps to investigate the use of push notifications in the wild and found that about half of these apps use Google's FCM for this purpose. We then conducted an in-depth analysis of 60 apps to determine whether these apps send sensitive data via push notifications and how they eventually protect this data. We found that most apps send some form of sensitive content, that there is neither a standard nor a uniform approach to protecting notification data, and that more than half of the apps analyzed use no protection at all.

Our work provides first insights into the end-to-end protection of notifications that software developers implement in their apps and is a basis for future work. Our results draw attention to the security of push notifications as they are used to transmit potentially sensitive information. Furthermore, our results show that the protection of notifications often does not take place, due to many factors such as the lack of appropriate standards, technologies, frameworks, and documentation. We have identified these shortcomings and developer-centric research gaps as major challenges on the road to wider adoption of end-to-end security in software.

AVAILABILITY

Our static analysis can be completely replicated using our list of APK hashes, the analysis script we used, and our app selection based on AndroZoo. For the qualitative study, we provide the name, version, and hashes of APK files (even from App Bundles), traffic captures, and screen recordings of our testing devices showing the exact behavior during examination. Some screen captures are blurred, and some traffic captures are omitted due to private data contained in the recording. The data is available as supplementary material.

TABLE 6. List of apps 30 that use clear-text FCM notifications.

Name	Category	Download	Sensitive Content
Instagram	social	5B+	username and text
imo	communication	1B+	username and text
Skype	communication	1B+	username and text
Snapchat	communication	1B+	username
TikTok	social	1B+	username, action and text
Truecaller	communication	1B+	text
Likee	social	500M+	username and text
LINE	communication	500M+	username and text
Pinterest	lifestyle	500M+	username and text
Azar	communication	100M+	username
Botim	communication	100M+	username and text
Discord	communication	100M+	username and text
Hago	social	100M+	username and text
Microsoft Teams	business	100M+	username and text
PayPal	finance	100M+	name, text and amount of money
Reddit	social	100M+	username and text
Simsimi	entertainment	100M+	username and text
Tango	social	100M+	username and text
Yahoo Mail	communication	100M+	username and subject
Zello	communication	100M+	username
ZEPETO	entertainment	100M+	username and text
BOSS Revolution	communication	10M+	username and text
Glide	communication	10M+	username and text
iSharing	communication	10M+	username and text
JusTalk	communication	10M+	username
Marco Polo	communication	10M+	username
myMail	communication	10M+	username and text
Voxer	communication	10M+	username and text
Rune	communication	5M+	username and text
TypeApp	communication	1M+	username and text

TABLE 7. FCM usage per age of the apps and categorized by month. The age is based on the first time an app was seen by the crawlers of the AndroZoo data set which has a lowest value of 2020-05 [42].

Month	Number of Apps	use FCM
2020-05	20963	61 %
2020-06	22099	58 %
2020-07	5135	46 %
2020-08	3285	45 %
2020-09	2702	45 %
2020-10	2303	46 %
2020-11	1976	46 %
2020-12	1909	48 %
2021-01	1418	45 %
2021-02	1094	45 %
2021-03	1702	49 %
2021-04	1541	51 %
2021-05	1474	49 %
2021-06	1288	47 %
2021-07	1329	46 %
2021-08	1433	52 %
2021-09	1156	48 %
2021-10	1330	52 %
2021-11	1223	56 %
2021-12	1267	53 %
2022-01	1050	50 %
2022-02	975	49 %
2022-03	1198	48 %
2022-04	1103	49 %
2022-05	1098	46 %
2022-06	1661	61 %
2022-07	1044	51 %
2022-08	1051	48 %
2022-09	905	50 %
2022-10	1302	44 %
2022-11	1506	45 %
2022-12	1469	47 %
2023-01	1162	47 %
2023-02	960	50 %
2023-03	1196	49 %
2023-04	734	48 %
2023-05	589	41 %
2023-06	716	49 %
2023-07	737	44 %
2023-08	633	51 %
2023-09	562	41 %
2023-10	498	38 %

APPENDIX A FIGURES

See Figures 9 and 10.

APPENDIX B TABLES

See Tables 5–7.

ACKNOWLEDGMENT

The authors would like to thank Dr. Kevin Allix from the University of Luxembourg for helping to add metadata to apps from the AndroZoo data set. Arvid Voigt provided valuable help developing preliminary work of the static and qualitative analysis methods. They also like to thank Jan Mohren, Bruce Falke, Mika Dörr, Florian Dehling, and Julius Roettger for their help in conducting the in-depth analysis of apps.

REFERENCES

[1] Google LLC. (Apr. 2024). *FCM Architectural Overview | Firebase Cloud Messaging*. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/fcm-architecture

[2] J. Lou, X. Zhang, Y. Zhang, X. Li, X. Yuan, and N. Zhang, "Devils in your apps: Vulnerabilities and user privacy exposure in mobile notification systems," in *Proc. 53rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Porto, Portugal, Jun. 2023, pp. 28–41. [Online]. Available: https://ieeexplore.ieee.org/document/10202604/

[3] R. Wyden. (Dec. 2023). *Wyden Letter To Department of Justice Regarding Smartphone Push Notification Surveillance*. [Online]. Available: https://www.wyden.senate.gov/imo/media/doc/wyden_smartphone_push_notification_surveillance_letter.pdf

[4] A. Couts and L. H. Newman. (Dec. 2023). *Police Can SPY on Your IOS and Android Push Notifications*. [Online]. Available: https://www.wired.com/story/apple-google-push-notification-surveillance/

[5] B. Schneier. (Mar. 2024). *Surveillance Through Push Notifications*. [Online]. Available: https://www.schneier.com/blog/archives/2024/03/surveillance-through-push-notifications.html

[6] Threema GmbH. (Dec. 2023). *Push Notifications and Data Privacy*. [Online]. Available: https://threema.ch/en/blog/posts/push-notifications-and-data-privacy

[7] Y. Jinishian. (Jun. 2018). *Thousands of Mobile Apps Leak Sensitive Data Via Misconfigured Firebase Backends*. [Online]. Available: https://www.linkedin.com/pulse/thousands-mobile-apps-leak-sensitive-data-via-yves-jinishian

[8] P. Verma and S. Patil, "Exploring privacy aspects of smartphone notifications," in *Proc. 23rd Int. Conf. Mobile Human-Computer Interact.*, Sep. 2021, pp. 1–13, doi: 10.1145/3447526.3472065.

[9] M. Backman. (Apr. 2014). *Push Messages Isn't Secure Enough*. [Online]. Available: https://medium.com/@BackmaskSWE/push-messages-isnt-secure-enough-69121c683cc6

[10] G. Hogben and M. Perera. (Jun. 2018). *Project Capillary: End-to-end Encryption for Push Messaging, Simplified*. [Online]. Available: https://android-developers.googleblog.com/2018/06/project-capillary-end-to-end-encryption.html

[11] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the Android ecosystem," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1184–1199, May 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8660581/

[12] X. Liu, S. Zhu, W. Wang, and J. Liu, "Alde: Privacy risk analysis of analytics libraries in the Android ecosystem," in *Security and Privacy in Communication Networks* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 198, R. Deng, J. Weng, K. Ren, and V. Yegneswaran, Eds., Cham, Switzerland: Springer, 2017, pp. 655–672, doi: 10.1007/978-3-319-59608-2_36.

[13] R. Zhou, M. Hamdaqa, H. Cai, and A. Hamou-Lhadj, "MobiLogLeak: A preliminary study on data leakage caused by poor logging practices," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, London, ON, Canada, Feb. 2020, pp. 577–581. [Online]. Available: https://ieeexplore.ieee.org/document/9054831/

[14] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! Assessing user data exposure to advertising libraries on Android," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2016, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/free-for-all-assessing-user-data-exposure-advertising-libraries-android.pdf

[15] W. Meng, R. Ding, S. P. Chung, S. Han, and W. Lee, "The price of free: Privacy leakage in personalized mobile in-app ads," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2016, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/price-of-free-privacy-leakage-personalized-mobile-in-app-ads.pdf

[16] W. Yang, Y. Zhang, J. Li, H. Liu, Q. Wang, Y. Zhang, and D. Gu, "Show me the money! Finding flawed implementations of third-party in-app payment in Android apps," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2017, pp. 1–15.

[17] S. Bojjagani, V. N. Sastry, C.-M. Chen, S. Kumari, and M. K. Khan, "Systematic survey of mobile payments, protocols, and security infrastructure," *J. Ambient Intell. Humanized Comput.*, vol. 14, no. 1, pp. 609–654, Jan. 2023, doi: 10.1007/s12652-021-03316-4.

[18] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system," in *Proc. 28th USENIX Secur. Symp.* Santa Clara, CA, USA, Aug. 2019, pp. 603–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon

[19] Y. Li, Z. Yang, Y. Guo, X. Chen, Y. Agarwal, and J. I. Hong, "Automated extraction of personal knowledge from smartphone push notifications," in *Proc. IEEE Int. Conf. Big Data*, Seattle, WA, USA, Dec. 2018, pp. 733–742.

[20] T. Li, X. Zhou, L. Xing, Y. Lee, M. Naveed, X. Wang, and X. Han, "Mayhem in the push clouds: Understanding and mitigating security hazards in mobile push-messaging services," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* USA: ACM, Nov. 2014, pp. 978–989, doi: 10.1145/2660267.2660302.

[21] B. F. Demissie and S. Ranise, "Assessing the effectiveness of the shared responsibility model for cloud databases: The case of Google's firebase," in *Proc. IEEE Int. Conf. Smart Data Services (SMDS)*, Chicago, IL, USA, Sep. 2021, pp. 121–131. [Online]. Available: https://ieeexplore.ieee.org/document/9592496/

[22] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? Uncovering the data leakage in cloud from mobile apps," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 1296–1310. [Online]. Available: https://ieeexplore.ieee.org/document/8835301/

[23] T. Liu, H. Wang, L. Li, G. Bai, Y. Guo, and G. Xu, "DaPanda: Detecting aggressive push notifications in Android apps," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, Nov. 2019, pp. 66–78. [Online]. Available: https://ieeexplore.ieee.org/document/8952509/

[24] Z. Xu and S. Zhu, "Abusing notification services on smartphones for phishing and spamming," in *Proc. 6th USENIX Workshop Offensive Technol.*, Bellevue, WA, USA, Aug. 2012, pp. 1–11. [Online]. Available: https://www.usenix.org/conference/woot12/workshop-program/presentation/Xu

[25] M. Wachs, Q. Scheitle, and G. Carle, "Push away your privacy: Precise user tracking based on TLS client certificate authentication," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Dublin, Ireland, Jun. 2017, pp. 1–9. [Online]. Available: http://ieeexplore.ieee.org/document/8002897/

[26] P. Loreti, L. Bracciale, and A. Caponi, "Push attack: Binding virtual and real identities using mobile push notifications," *Future Internet*, vol. 10, no. 2, p. 13, Jan. 2018. [Online]. Available: https://www.mdpi.com/1999-5903/10/2/13

[27] S. A. Kollmann and A. R. Beresford, "The cost of push notifications for smartphones using tor hidden services," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Apr. 2017, pp. 76–85. [Online]. Available: http://ieeexplore.ieee.org/document/7966975/

[28] C. Schindler, M. Atas, T. Strametz, J. Feiner, and R. Hofer, "Privacy leak identification in third-party Android libraries," in *Proc. 7th Int. Conf. Mobile Secure Services*, Gainesville, FL, USA, Feb. 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9727217/

[29] Y. Chen, T. Li, X. Wang, K. Chen, and X. Han, "Perplexed messengers from the cloud: Automated security analysis of push-messaging integrations," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1260–1272, doi: 10.1145/2810103.2813652.

[30] J. Smith, L. N. Q. Do, and E. Murphy-Hill, "Why can't Johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *Proc. 16th Symp. Usable Privacy Secur.*, Aug. 2020, pp. 221–238. [Online]. Available: https://www.usenix.org/conference/soups2020/presentation/smith

[31] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, M. Kantarcioglu, and D. Yao, "CryptoGuard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., Nov. 2019, pp. 2455–2472, doi: 10.1145/3319535.3345659.

[32] I. Ryan, U. Roedig, and K.-J. Stol, "Unhelpful assumptions in software security research," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 3460–3474, doi: 10.1145/3576915.3623122.

[33] A. Herzberg and H. Leibowitz, "Can Johnny finally encrypt: Evaluating E2E-encryption in popular IM applications," in *Proc. 6th Workshop Socio-Tech. Aspects Secur. Trust*, Los Angeles, CA, USA, Dec. 2016, pp. 17–28, doi: 10.1145/3046055.3046059.

[34] S. Dechand, A. Naiakshina, A. Danilova, and M. Smith, "In encryption we don't trust: The effect of end-to-end encryption to the masses on user perception," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Stockholm, Sweden, Jun. 2019, pp. 401–415. [Online]. Available: https://ieeexplore.ieee.org/document/8806742/

[35] R. Abu-Salma, E. M. Redmiles, B. Ur, and M. Wei, "Exploring user mental models of end-to-end encrypted communication tools," in *Proc. 8th USENIX Workshop Free Open Commun. Internet*, Baltimore, MD, USA, Aug. 2018, pp. 1–8. [Online]. Available: https://www.usenix.org/conference/foci18/presentation/abu-salma

[36] C. Stransky, O. Wiese, V. Roth, Y. Acar, and S. Fahl, "27 years and 81 million opportunities later: Investigating the use of email encryption for an entire university," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2022, pp. 860–875. [Online]. Available: https://ieeexplore.ieee.org/document/9833755/

[37] 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 *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2012, pp. 50–61, doi: 10.1145/2382196.2382205.

[38] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To pin or not to pin—Helping app developers bullet proof their TLS connections," in *Proc. 24th USENIX Secur. Symp.*, Washington, DC, USA, Aug. 2015, pp. 239–254. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/oltrogge

[39] M. Tahaei and K. Vaniea, "A survey on developer-centred security," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Stockholm, Sweden, Jun. 2019, pp. 129–138. [Online]. Available: https://ieeexplore.ieee.org/document/8802434/

[40] M. Gutfleisch, J. H. Klemmer, N. Busch, Y. Acar, M. A. Sasse, and S. Fahl, "How does usable security (Not) end up in software products? Results from a qualitative interview study," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2022, pp. 893–910. [Online]. Available: https://ieeexplore.ieee.org/document/9833756/

[41] H. Assal and S. Chiasson, "'Think secure from the beginning': A survey with software developers," in *Proc. CHI Conf. Human Factors Comput. Syst.*, Glasgow Scotland, U.K., May 2019, pp. 1–13, doi: 10.1145/3290605.3300519.

[42] K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR)*, Austin Texas, May 2016, pp. 468–471, doi: 10.1145/2901739.2903508.

[43] Google LLC. (2024). *Firebase Cloud Messaging*. [Online]. Available: https://firebase.google.com/docs/cloud-messaging

[44] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang, "Understanding Android obfuscation techniques: A large-scale investigation in the wild," in *Security and Privacy in Communication Networks* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 254, R. Beyah, B. Chang, Y. Li, and S. Zhu, Eds., Cham, Switzerland: Springer, 2018, pp. 172–192, doi: 10.1007/978-3-030-01701-9_10.

[45] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing* (Springer Topics in Signal Processing), vol. 2. Berlin, Germany: Springer, 2009, pp. 1–4, doi: 10.1007/978-3-642-00296-0_5.

[46] A. Cortesi, M. Hils, and T. Kriechbaumer. (2010). *Mitmproxy: A Free and Open Source Interactive HTTPS Proxy*. [Online]. Available: https://mitmproxy.org/

[47] Google LLC. (Dec. 2023). *FCM Ports and Your Firewall | About FCM Messages | Firebase Cloud Messaging*. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/concept-options#messaging-ports-and-your-firewall

[48] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, *The Messaging Layer Security (MLS) Protocol*, document 9420, Jul. 2023, p. 132. [Online]. Available: https://www.rfc-editor.org/info/rfc9420

[49] A. Carboneri, M. Ghasemisharif, S. Karami, and J. Polakis, "When push comes to shove: Empirical analysis of web push implementations in the wild," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Austin TX USA, Dec. 2023, pp. 44–55, doi: 10.1145/3627106.3627186.

[50] M. Thomson, *Message Encryption for Web Push*, document RFC8291, Nov. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8291

THOMAS NETELER received the B.Sc. and M.Sc. degrees in media technology from the TH Köln University of Applied Sciences, Cologne, Germany.

He is currently a Researcher with the Institute for Cyber Security and Privacy (ICSP), H-BRS University of Applied Sciences, Sankt Augustin, Germany. His research interest includes developer-centered security, with a focus on the usability of end-to-end security technologies for developers.

SASCHA FAHL received the Diploma (equivalent to the Master of Science) degree in computer science Philips University Marburg, Germany, in 2010, and the Dr.rer.nat. (equivalent to the Ph.D.) degree in computer science from Leibniz University Hannover, Germany, in 2016.

He is currently a tenured Faculty Member with the CISPA Helmholtz Center for Information Security and a Full Professor of empirical information security with Leibniz University Hannover. Previously, he was a Professor at Ruhr University Bochum, Germany, and the Chair of information security at Leibniz University Hannover. He was an Independent Research Group Leader at CISPA, Saarland University. He worked at the Chrome Security Team. He was a Researcher at Fraunhofer FKIE, Bonn.

Prof. Fahl research won the Distinguished Paper Award at IEEE Security and Privacy, the Best Student Paper Award at IEEE Security and Privacy, the NSA's Best Scientific Cybersecurity Paper Competition, and the Google Faculty Research Award. He is a recipient of the Heinz Maier-Leibnitz Prize and the "Curious Mind" Award of the Manager Magazin. His web page is https://saschafahl.de.

LUIGI LO IACONO received the Dipl.-Ing. (equivalent to the Master of Science) and Dr.-Ing. (equivalent to the Ph.D.) degrees in computer science from the University of Siegen, Germany, in 1999 and 2005, respectively.

He is currently a Professor of information security with the H-BRS University of Applied Sciences, Sankt Augustin, Germany, where he is the Co-Founder and the Heads of the Institute for Cyber Security and Privacy (ICSP). His research interest includes security and privacy-enhancing technologies for distributed software systems with a particular focus on their usability.