No Privacy Among Spies: Assessing the Functionality and Insecurity of Consumer Android Spyware Apps

Enze Liu
UC San Diego
La Jolla, CA, USA
e7liu@eng.ucsd.edu

Sumanth Rao
UC San Diego
La Jolla, CA, USA
svrao@ucsd.edu

Sam Havron*
Cornell Tech
New York, NY, USA
havron@cs.cornell.edu

Grant Ho
UC San Diego
La Jolla, CA, USA
grho@eng.ucsd.edu

Stefan Savage
UC San Diego
La Jolla, CA, USA
savage@cs.ucsd.edu

Geoffrey M. Voelker
UC San Diego
La Jolla, CA, USA
voelker@cs.ucsd.edu

Damon McCoy
New York University
Brooklyn, NY, USA
mccoy@nyu.edu

ABSTRACT

Consumer mobile spyware apps covertly monitor a user's activities (i.e., text messages, phone calls, e-mail, location, etc.) and transmit that information over the Internet to support remote surveillance. Unlike conceptually similar apps used for state espionage, so-called "stalkerware" apps are mass-marketed to consumers on a retail basis and expose a far broader range of victims to invasive monitoring. Today the market for such apps is large enough to support dozens of competitors, with individual vendors reportedly monitoring hundreds of thousands of phones. However, while the research community is well aware of the existence of such apps, our understanding of the mechanisms they use to operate remains ad hoc. In this work, we perform an in-depth technical analysis of 14 distinct leading mobile spyware apps targeting Android phones. We document the range of mechanisms used to monitor user activity of various kinds (e.g., photos, text messages, live microphone access) — primarily through the creative abuse of Android APIs. We also discover previously undocumented methods these apps use to hide from detection and to achieve persistence. Additionally, we document the measures taken by each app to protect the privacy of the sensitive data they collect, identifying a range of failings on the part of spyware vendors (including privacy-sensitive data sent in the clear or stored in the cloud with little or no protection).

KEYWORDS

Android Spyware, Android Security, Consumer Spyware Apps, Reverse Engineering, Android API Abuse

*Work done while this author was affiliated with Cornell Tech.

Proceedings on Privacy Enhancing Technologies 2023(1), 1–18

https://doi.org/https://doi.org/10.56553/popets-2023-0001

1 INTRODUCTION

Consumer mobile spyware — software that covertly gathers information on a mobile device and transfers that information to a remote server — has existed for at least two decades, but has grown significantly in popularity in recent years. In one recent study from Norton Labs [1], the number of devices identified with spyware apps increased by 63% between September 2020 and May 2021. A similar report from Avast saw a 93% increase in the use of spyware apps in the UK over a similar period [2].

Sold under a wide variety of brand names — TheTruthSpy, mSpy, Flexispy and so on — these apps are marketed directly to the general public. They are relatively cheap (typically between $30 and $100 per month), easy to install and do not require specialized technical know-how to deploy or operate. Indeed, the only requirements for such software is *temporary* physical access to the target device and the ability to install an off-store app.[1] After installation, the owner of the target device may have no knowledge that anything has changed. But the intimate details of their life can now be sent to another party, including the contents of their text messages, email messages, photos taken and received, and even live recording from their microphone and camera. Unsurprisingly, such "stalkerware" has been implicated in a range of abuses including intimate partner violence [3] and cyberstalking [4].

Moreover, privacy failures in the "back end" software used to store and display exfiltrated data means that exposure of the victims' private information is not limited to just the abuser who installs the software, but also to miscreants who exploit the insecure design and/or implementations of these apps. Indeed, a plethora of reports indicate that a broad range of consumer spyware cloud services have been breached, exposing hundreds of thousands (if not millions) of users' private data [5–14].

However, while the existence of such software, and the threat it poses, is well documented in mass media, the technical methods that these apps use to pervasively mine and exfiltrate private data

[1]Because their features violate store policy, app stores like Google Play do not permit the sale of popular spyware apps.

is not well understood. How does such software hide on the target device? How does it acquire the contents of text messages or of third-party applications? How do these apps monitor a victim's camera or microphone without notifying the user? Are there a small number of common techniques for bypassing protections or does each vendor innovate independently? Understanding these issues, as well as the nature of the cloud services used to store the most sensitive data captured from target devices, is the motivation for this work.

In particular, our paper describes a broad technical investigation into 14 leading consumer spyware apps for Android-based smart phones. Specifically, we seek to answer two key questions:

- How do spyware apps achieve their advertised functionalities? We focus on stealthy features that facilitate nonconsensual tracking.
- What are the measures taken by spyware apps to protect the data they collect?

To address these questions empirically, we reverse engineered 14 of the most popular consumer spyware apps. In analyzing their behavior we make three primary contributions:

- We performed the first comprehensive and in-depth analysis of mechanisms used by consumer spyware apps to bypass or trick system level isolation across a range of different feature categories.
- In addition to confirming the broad use of some techniques identified in the mobile malware literature (e.g., abusing "accessibility" APIs), we identified two novel abuses of Android APIs (new techniques of invisible camera access and hiding app icons). We also document that Android's threat model does not include abuses of their APIs that allow apps to conceal their icon.
- We tested spyware apps *in situ* in a carefully monitored environment and analyzed their communications with the cloud service components. We believe our work is also the first academic effort to document in detail a range of privacy deficiencies (e.g., data sent in plaintext and cloud services with insecure direct object references (IDOR [15, 16]) for the contents of phone data).

Together, we believe that this work further sharpens the community's understanding of consumer mobile spyware, providing guidance both for phone OS vendors and regulators in their efforts to mitigate the use and availability of such software.

2 SPYWARE

To provide context for our study, we first describe how spyware is used in general and then describe how we selected the specific spyware apps we investigate.

2.1 Spyware Use

Spyware enables an adversary to surreptitiously record the activities, behavior, and location of a victim based on the victim's phone usage. These invasive apps have a wide range of capabilities for monitoring the victim, and do so using existing Android APIs without needing root access.

To install these apps, the adversary first creates an account on the spyware's web portal interface and purchases a subscription, if required. The adversary then requires physical access to the victim's device to download and install the spyware app. When installing, the app requests an extensive set of Android permissions as the basis for performing its monitoring activity. The adversary readily grants these permissions to the app, a step which entirely undermines the Android app permission model. The adversary then signs in to the app on the device, typically using the same credentials used by the web portal, and links the victim's device with the adversary's account on the spyware portal.

Subsequently, the adversary no longer needs physical access to the victim's device, and can remotely control the actions of the app via the online spyware portal. The spyware app both passively collects data on the victim's device (e.g., location, text messages, calls, etc.), and also allows the adversary to perform on-demand remote actions like covertly recording audio and video, taking app screenshots, etc.

Although spyware authors exhibit tremendous effort and creativity in subverting Android sandbox protections to implement their capabilities, many apps spend much less effort in protecting the data that they exfiltrate and store on their backend servers. Later, in Section 4 we describe several vulnerabilities in spyware apps that put the victim's data at risk to a third-party attacker.

2.2 Spyware App Selection

For our analysis, we chose 14 distinct leading consumer Android spyware apps.[2] We focus on Android-based spyware because most of the mobile spyware market appears to be focused there. Since curated app stores like Google Play do not permit the sale of such apps, in practice they must be side-loaded off-store, a process that Apple does not support. As a result, consumer mobile spyware only operates on "rooted" iPhones. Rooting an iPhone can be a technically involved operation (one popular guide to jailbreaking the iPhone involves 41 distinct steps [17]) and one that can take significant time to complete — both requirements at odds with the broad, non-technical customer base such apps are marketed to. We also focus on leading spyware apps as they are the apps that more people are exposed to and they are more likely to be innovative (new features could potentially bring them more customers).

In our selection process, we started with the 18 off-store spyware apps identified by Chatterjee et al. [3]. We augmented our set of apps by taking the intersection of two major industry reports ([18] and [19]) that listed spyware apps.[3] Next, we sorted all the apps based on the Tranco ranking [20] of their website domain (snapshot taken on May 5th, 2022). We filtered out apps that were distributed via Google Play (since, to appear in the store, they do not have compelling spyware capabilities) or broken (e.g., not reachable or no longer accepting payments). For apps that are rebranded versions of other apps, we consider them duplicates and did not examine them separately (Appendix A.2 details how we detect duplicate apps).

[2] Often the same spyware app is available under multiple names. Appendix A.2 describes how we identify and avoid choosing duplicate apps.

[3] We take the intersection of both reports because (a) the definition of consumer spyware is vague; and (b) the collection process is ad-hoc and differs for each report. Apps in the intersection represent consensus among these disparate classifications.

App Name	Website Domain	Ranking	Portal Domain	Ranking	Target SDK	Package Name
mSPY	mspy.com	46k	mspyonline.com	220k	25	core.update.framework
Mobile-tracker-free	mobile-tracker-free.com	54k	mobile-tracker-free.com	54k	28	mobile.monitor.child2021
Clevguard	clevguard.com	69k	clevguard.com	69k	28	com.kids.pro
HoverWatch	hoverwatch.com	87k	hoverwatch.com	87k	28	com.android.core.mntw
Flexispy	flexispy.com	107k	flexispy.com	107k	22	com.fp.backup
Spyic	spyic.com	152k	spyic.com	152k	22	com.sc.spyic.v3
Spyhuman	spyhuman.com	179k	spyhuman.com	179k	22	m.mobile.control
TheTruthSpy	thetruthspy.com	214k	thetruthspy.com	214k	28	com.systemservice
iKeyMonitor	ikeymonitor.com	230k	emcpanel.com	1.1m	23	com.sec...im20190419*
Cerberus	cerberusapp.com	251k	cerberusapp.com	251k	23	com.lsdroid.cerberus
Spy24	spy24.app	284k	spy24.net	2.4m	29	net.spy24.wifi
Spapp	spappmonitoring.com	421k	spappmonitoring.com	421k	26	com.monspap.alarm
Meuspy	meuspy.com	485k	meuspy.com	485k	32	br.com.sistema.aplicativo
Highstermobile	highstermobile.com	590k	evt17.com	1.5m	30	org.secure.smsgps

Table 1: The 14 spyware apps we study, their website domain and its corresponding Tranco ranking, their portal domain and its corresponding Tranco ranking, and their APK's target SDK version and package name. Tranco rankings taken on May 5th, 2022. Shading added to improve readability. *iKeyMonitor's full package name is 'com.sec.android.internet.im.service.im20190419'.

From the top 25 most popular apps, we identified 14 distinct apps which we used in our study. The apps in this set are produced by a wide spectrum of vendors, with a broad array of capabilities, and cover the majority of apps identified in prior work [3] (10 out of the 14 apps that are still alive). Moreover, we find significant overlap in the low-level technical implementation of spyware capabilities: all but three techniques described in Section 3 are discovered after reverse engineering the top five most popular apps (in terms of website domain ranking), and no new techniques are found after the 11th ranked app (Spy24).[4] The lack of new techniques discovered among less popular apps in our set suggests that the apps we study capture a representative set of techniques used in practice.

Table 1 shows the list of spyware apps we chose, their website domain and its corresponding Tranco ranking, their web portal domain and its corresponding Tranco ranking, and their APK's target SDK version and package name. Table 6 in Appendix C provides additional information (version code and SHA-1 hash) about their APK files.

3 SPYWARE ABUSE OF ANDROID APIS

In this section, we explain how spyware apps implement various privacy invasive capabilities using existing APIs supported by the Android OS. We start by discussing our methodology for identifying these capabilities and uncovering their associated implementations. We then summarize the set of basic capabilities (§ 3.2) that either do not require any permissions or are enabled just by acquiring permissions, and group all of the other capabilities that we study into three categories based on their goals: stealthily collecting a victim's information (§ 3.3), hiding the app's presence on the phone (§ 3.4), and persistently spying over a long period of time (§ 3.5). For each category, we describe each capability in the category and its associated implementations, and end the category by discussing potential mitigations.

3.1 Methodology

We gather a list of privacy invasive capabilities from two sources: (1) capabilities advertised and offered by spyware apps; and (2) prior academic papers and industry reports described in Section 5 that look at how the Android API can be used to achieve such unintended functionality. Table 2 summarizes the capabilities we study and shows which spyware apps implement them. To understand how the spyware apps implement their nefarious capabilities, we reverse engineered and analyzed each app. Our approach consists of three steps.

Preparation. We acquire the latest APKs of spyware apps by directly downloading them from their websites. We use `APKTOOL` [21] to disassemble the APKs and obtain the corresponding manifest and smali code.[5] We also decompile the APK to Java source using `JADX` [22]. Identifiers and even strings in the Java code are often obfuscated, and Appendix B provides more details about the obfuscation used by various apps.

Manual Investigation. For each capability, we manually analyzed the app manifest and the decompiled Java code to determine how each capability was implemented using the standard Android API: we manually located the API primitives associated with each capability, examined the relevant control flow, and confirmed that the primitives we identified were reachable. While we primarily used the decompiled Java code for analysis, we reviewed the smali code when necessary (e.g., when a function fails to decompile). Since the variable and method names were often obfuscated, we renamed them to human-readable ones as we examined the decompiled Java code (our naming annotations are available upon request).

[4]While we do continue to find different variants or implementation of the same technique, we view these as minor details.

[5]Smali is a human-readable represenation of the Dalvik bytecode used by Android apps.

Category	Capabilities	mSPY	Mobile-tracker-free	Cleyguard	HoverWatch	Flexispy	Spyic	Spyhuman	TheTruthSpy	iKeyMonitor	Cerberus	Spy24	Spapp	Meuspy	Highstermobile
Basic Capabilities (§ 3.2)	Ambient Recording		★			★		★	★	★	★	★	★	★	
	Calendar	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Call Logs	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Clipboard		★						★	★		★			
	Contacts	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Info of Other Applications	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Location	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Network Info	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Phone Info	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	SMS or MMS	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Shared Media Files	★	★	★	★	★	★	★	★	★	★	★	★	★	★
Data Gathering (§ 3.3)	Invisible camera access		★	★	★	★		★	★	★	★	★	★	★	★
	Invisible microphone access		★	★	★	★		★	★	★		★	★	★	
	Accessing protected data	★	★	★	★	★	★	★	★	★	★	★	★	★	★
	Taking screenshots	★	★	★	★			★		★		★	★	★	
Hiding the App (§ 3.4)	Hiding app icon	★	★	★	★	★	★	★	★	★	★	★	★	★	
	Launching a hidden app		★		★	★	★	★	★	★	★	★	★	★	
	Hide from recents screen	★	★	★	★	★	★	★		★	★	★		★	★
Persistence (§ 3.5)	Obscuring the uninstallation process	★	★	★		★		★	★	★	★	★	★	★	
	Creating "diehard" services	★	★	★	★	★	★	★	★	★	★	★	★	★	★

Table 2: Summary of capabilities studied. A star denotes that an app implements a particular capability.

Once we discovered an implementation for a particular capability and the API primitives one app used, we expedited our search process for other apps by searching for use of the same API primitives, and manually reviewed the results to remove false positives. If this API similarity approach did not yield results, we reverted back to manually inspecting the code.

We use two heuristics to guide our manual investigation process. First, many apps can receive remote commands (e.g., using third-party libraries such as Firebase [23]). The method responsible for dispatching remote commands is a useful top-down starting place for identifying the underlying implementation (e.g., following the methods it calls in response to remote commands). For apps that do not receive remote commands, they operate as background services to covertly collect user data. These background services often are responsible for dispatching various tasks and contain useful leads. Second, for any capability, there are limited ways to implement it using Android API primitives, and many of them are publicly documented (e.g., capturing audio through `MediaRecorder`). Searching for these API primitives in the code helps to quickly locate methods associated with a capability. Additionally, once we identified an implementation of a capability, we examined its control flow in a bottom-up fashion to identify the operation that would trigger its execution (typically leading back to dispatch methods).

Confirmation. For each implementation of a capability (and related API primitives) we discover, we verified our analysis conclusions by developing an app using the same API primitives and testing it on a Pixel 4 phone running Android 11. We also installed and ran each spyware app to ensure the capabilities we observed were indeed enabled. Since the majority of the apps we study have been implemented using Android APIs up to Android 11, we discuss their implementations in that context. However, we also discuss the impact changes in Android 12 can have on the spyware apps.

Code Release. Our code that implements all the API primitives discovered in this paper is available upon request. Similarly, the annotated class and variable names are available upon request (which can be used to reproduce our results after independently acquiring the corresponding APK files). To avoid potential copyright infringement risks, we do not provide the decompiled and annotated source code itself.

3.2 Basic Capabilities

As seen in Table 2, all spyware apps support a large, common set of basic capabilities for collecting sensitive user data. These capabilities either do not require any permission (e.g., clipboard) or are easily enabled after obtaining particular Android permissions (e.g., reading SMS messages, contacts, call logs, etc.). Spyware apps can obtain these capabilities simply by requesting the associated permissions in the manifest at installation or their first run. Additionally, by design most of these capabilities (except for location and ambient recording) do not provide any notification to the user and are straightforward to implement.

3.3 Data Gathering

Spyware apps seek to stealthily collect a victim's information without being noticed. In this section, we describe how they covertly access the camera, the microphone, and protected data of other apps, as well as how they take screenshots all without the victim's knowledge.

Capability #1: Invisible Camera Access. Spyware apps use a device's camera to take pictures, record videos, and stream live videos. To remain hidden from the victim, spyware apps need to access the camera without being noticed. The apps we studied use three methods to achieve surreptitious camera access: (1) using an invisible preview; (2) intercepting raw frames without a preview; and (3) using an invisible browser. While the first technique is well-documented in the malware literature (e.g., Yan et al. [24]), the second and third techniques have not been reported before and reflect the creativity that spyware authors continue to apply in abusing the Android API to implement their capabilities.

Invisible Preview. One way to use the camera is to create a preview that displays an image or video to users. Once a preview is started, apps can take pictures or record videos. Benign apps (such as selfie apps) use a preview so that users can see the current status of the camera. Spyware apps, in contrast, seek to hide this preview by setting the preview to an invisible size (typically a 1x1 pixel) or making the preview transparent.

Raw Frames Without a Preview. Spyware apps can also access the output of the camera without displaying a preview using advanced camera features supported by Android [25]. For example, apps can capture raw frames from the camera with either an `ImageReader` [25] or `SurfaceTexture` [26], neither of which requires showing a preview on the screen. Benign apps (such as photo apps) use these features for advanced operations such as frame-by-frame processing. Spyware apps, on the other hand, use these features to capture raw frames, either saving the frames locally or streaming the frames to a remote server, all without the victim noticing. While the expectation for such advanced features is that the processed frames will be displayed to users eventually (which is normally the case for benign apps), this by no means is enforced. Finally, we note that four apps rely on the third-party library WebRTC (or its variants) to achieve this functionality (WebRTC uses both `ImageReader` and `SurfaceTexture`).

Invisible Browser. Unique to Spy24, this app uses an invisible `WebView` [27] (an in-app browser) with a 1x1 pixel size for streaming live videos. While the browser may be invisible, the app can stream full camera resolution to a Spy24 server. `WebView` allows users to browse web content in an app while providing advanced configuration options to developers. Spy24, in this instance, programmatically sets the user agent, enables JavaScript, and disables the cache. This configuration flexibility provided by `WebView` is necessary for streaming via a web browser.

Once the browser is configured, it accesses the camera via a JavaScript version of WebRTC (loaded dynamically) and streams live video back to a Spy24 server. This approach evades static analysis that examines API calls in the APK file, as the JavaScript file is loaded during runtime. Listing 1 shows a snippet of code that demonstrates how Spy24 streams video with an invisible browser.

```
// xml layout for the webview
<WebView id="@+id/wv" layout_width="1dp"
    layout_height="1dp"/>
// Java code that configures the webview
WebView webView = findViewById(R.id.wv);
webView.getSettings().setUserAgentString("
    some_user_agent");
// Javascript code that activates the camera when
    loaded by the webview
userAgent.includes('some_user_agent'){
  camera.start()
}
```

Listing 1: Code snippet that demonstrates how Spy24 streams live video.

First, it adds the 1x1 `WebView` as an overlay. It then sets a specific user agent to act as a trigger. Upon contacting the server, the `WebView` loads a JavaScript file that activates the camera if the `WebView` has the specified user agent.

Capability #2: Invisible Microphone Access. Spyware apps use the microphone to covertly record ambient sound, phone calls, and voice calls (calls made in third-party apps like WhatsApp). Ambient recording is enabled after acquiring the Android permission to record; below we focus on how apps achieve phone call recording and voice call recording.

Phone call recording on Android has evolved over time. Prior to Android 6, apps could easily use `MediaRecorder` to record phone calls [28]. Specifying `MediaRecorder.AudioSource.VOICE_CALL`, an app can receive both uplink and downlink audio for a call. Since Android 6, `VOICE_CALL` is protected by a permission reserved for system apps and cannot be requested by third-party apps [29]. Spyware apps (and other recording apps) use native code via the Android NDK to circumvent this limitation, specifying `VOICE_CALL` by calling functions defined in `libmedia.so` [30]. Android 9 subsequently prevented this workaround by filtering out unauthorized attempts to set `AudioSource` [31, 32].

After Android 9, spyware apps started recording phone calls using Microphone, which nominally only captures uplink audio. However, spyware apps employ several workarounds to also capture downlink audio. For instance, some spyware apps turn on the speaker and turn off noise-cancelling during a phone call so that the microphone can potentially capture downlink audio via the speaker.

Voice call recording — recording calls made in third-party apps such as WhatsApp — became possible with Android 10 for apps that register as an accessibility service. Prior to Android 10, recording voice calls was not possible, as Android only allowed one app to access input audio at a time [33]. Android 10 relaxed this restriction, allowing two apps to access input audio concurrently if one of them is an `AccessibilityService`. This feature was intended to allow an accessibility service to perform voice recognition, however spyware apps quickly abused it. To detect that a voice call is active or a voice message is being played, spyware apps use side-channels such as listening for specific notifications triggered by voice calls or looking for specific actions (e.g., users clicking on the OK button to start the voice call) on the screen via the accessibility interface.

Capability #3: Accessing Protected Data. A core feature of spyware apps is the ability to collect data that they are not supposed to read, which includes reading protected data of other apps (e.g., WhatsApp messages) and recording keystrokes to other apps. Accessing this data is normally impossible as each app runs in its own sandbox. Spyware apps achieve these capabilities primarily by abusing the accessibility permission which does not require rooting, echoing prior literature [34–40] that investigates how accessibility can be abused.

Reading protected data of other apps can be performed in two ways: through accessibility or notification. Most academic literature has focused on accessibility, and only a few reports [41] have examined the use of notification. Accessibility services are expected to read what is rendered on the screen as they are designed to help disabled and impaired users. Spyware apps abuse this capability to read information displayed by other apps and extract sensitive information when an app (e.g., WhatsApp) is used.

Notification, on the other hand, allows spyware apps to read sensitive information even if the spyware app is not active. While reading from a notification has limitations (e.g., a spyware app cannot read images), nine spyware apps abuse it. Finally, we note that, like accessibility, both benign apps and malicious apps make use of notifications as a side-channel for accessing information: benign apps (an example is [42]) do so to avoid triggering the "Read Receipt" feature, which notifies the sender if the recipient has read the message, of other apps (e.g., Signal and WhatsApp), while malicious apps do so purely to extract information.

Recording keystrokes is done via accessibility. Prior work [34, 36, 40] has demonstrated how the accessibility framework in Android can be abused for recording keystrokes. Spyware apps simply use the `getText` method, part of Android's accessibility framework, to track all key presses from the user in any app. While there are built-in mechanisms to prevent `getText` from reading sensitive information such as credentials (e.g., by setting the attribute `importantForAccessibility` to false), past work [40] has shown that these defenses are not widely adopted. Additionally, mitigations like setting `importantForAccessibility` to false also reduces the usability of benign apps such as screen readers.

Capability #4: Taking Screenshots. Spyware apps can also extract sensitive information by taking screenshots. We observe two ways the apps in our set take screenshots: via `MediaProjection` and `AccessibilityService`. For MediaProjection to work, an activity is required [43]. This approach is nominally a challenge for spyware apps since they generally run as services and do not have an activity. As a result, some spyware apps create temporary activities that are transparent and not visible to users. These invisible activities are then removed after the screenshot is taken. Android 10 limited apps that can start activities to a few scenarios [44] (e.g., if an app is an accessibility service or has the `SYSTEM_ALERT_WINDOW` permission). Such restrictions make it harder for spyware apps to use `MediaProjection` to take screenshots, but still leaves a loophole as spyware apps can register as an accessibility service and request the `SYSTEM_ALERT_WINDOW` permission.

However, Android 11 (and later) has made it simple for spyware apps to take screenshots by adding the `takeScreenshot` API to `AccessibilityService` [45]. This method requires no activity and is easy for spyware apps to implement.

3.3.1 Mitigations. Android has used a variety of approaches to mitigate some of the issues mentioned in this section.

The first approach is to pose additional restrictions on apps that seek to access certain information. One type of restriction is to require apps to be in a certain state: Android 10 restricted access to the clipboard information to apps that are either the default input method editor or is the app that currently has focus [46]. As a result of this patch, none of the spyware apps were able to acquire clipboard information[6]. Another type of restriction is to require apps to possess additional permissions: besides requiring the permission to record audio, Android 10 further demands an app to be an accessibility service to capture audio input during phone calls. However, we note that permission-based restrictions are unlikely to be successful, as the abuser has physical access to the device and can grant arbitrary permissions.

A second approach is to provide a visible indicator to the user that cannot be dismissed while a granted permission is being used by any app. For example, Android has a visible persistent indicator for location and screen recording use, and Android 12 introduced a visible indicator for microphone and camera use. Such an indicator confirms to the user when they expect it to be used, and alerts the user when they do not.

This approach can also fail depending on the use case. For example, a persistent indicator is useful to alert users to long-lasting actions such as recording. However, a persistent indicator is unlikely to be helpful for sub-second operations (e.g., taking a screenshot and taking a picture), and users can sometimes miss these indicators (e.g., if the phone is in a pocket or purse). Moreover, benign use cases will also trigger persistent indicators, which can then mask simultaneous spyware activity. For example, the indicator for the microphone will be displayed when a call is ongoing regardless of whether a spyware app is performing call recording in the background or not. In this case, spyware apps can safely record calls in the background, as the indicator is always on. Indeed, sophisticated spyware can reduce suspicion by carefully timing their actions (e.g., only performing an action when the associated indicator is already triggered by another app).

A third approach is to provide a convenient way for users to review the permissions that have been granted to all installed apps. Android 12 takes this approach by introducing a privacy dashboard, which displays the accesses in the last 24 hours to sensitive information such as location, microphone, and camera. While such a dashboard does require users to proactively take action, it is a potentially useful tool for users to quickly identify suspicious apps based upon the plethora of invasive permissions they have been granted. We recommend that all actions to access sensitive data should be added to the privacy dashboard and users should be periodically notified of the existence of apps with excessive number of permissions.

[6] We are aware of a workaround [47] that can circumvent this restriction. However, it requires extensive user interaction and is not adopted by any of the spyware apps. We also note that iKeyMonitor attempted to evade this restriction by trying to acquire focus with an invisible `EditText` of size 1x1. However, this attempt appears to be unsuccessful: both iKeyMonitor and our cloned implementation fail to read the clipboard.

Lastly, Android also employs various mechanisms that alert users of sensitive actions or potentially malicious activities. The most straightforward example is the deployment of Google Play Protect, which warns users when a potentially malicious app is being installed. As well, Android 9 requires a persistent notification for background apps that seek to access the camera and microphone: these apps need to run as a foreground service, which triggers a persistent notification that cannot be dismissed by the apps themselves. Finally, Android uses audio cues (e.g., a shutter sound) to notify users of operations such as audio and video recording. Sadly, we note that all these mechanisms can fail, as they can be turned off either programmatically (e.g., audio cues normally can be turned off after acquiring permissions to manipulate audio settings[7]) or by a malicious user (e.g., as mentioned in Appendix A.1, spyware apps require that their notification be disabled by the abuser upon installation).

The research community has also proposed various defense mechanisms when studying some of issues mentioned in this section. Huang et al. [39], for instance, seek to constrain accessibility misuse with the assumption that the user is benign, which unfortunately does not hold in the context of spyware apps. Another example is Petracca et al. [49], who proposed requiring user approval before an app can record to stop malicious apps from eavesdropping. However, we note that this defense is unlikely to be successful as the adversary generally has physical access to the target device. On the one hand, this defense will only work if user approval is requested every time, which leads to significant usability issues. On the other hand, if user approval is only requested once when an app tries to record, it is unlikely to be successful as the abuser will have physical access to the phone and can grant the consent. Lastly, Yan et al. [24] developed a system for detecting overlay-based Android malware. However, their system is deployed on an Android app store and does not defend against spyware apps that are sideloaded.

3.4 Hiding the App

In addition to surreptitiously spying on victims, spyware apps also use a variety of methods to keep the existence of the app hidden from the victims. These tactics include hiding the app icon from the app launcher, launching a hidden app, and hiding from the Recent screen.

Capability #5: Hiding App Icon. To hide their presence, spyware apps hide their app icons from the app launcher that lists the available apps [50]. Figure 1 shows an example of the app launcher, and 13 of the 14 spyware apps in our study include code to hide their app icons from the app launcher using two methods: (1) hiding the app icon at runtime, and (2) specifying no launcher activity.

Hiding the app icon at runtime can be implemented by invoking the method `setComponentEnabledSetting()` and disabling the launcher activity. Previous work [51] examined hiding app icons in this way, and this approach works on devices that run Android 9 or below. An activity of category `LAUNCHER` indicates that it should be displayed in the app launcher [52], and disabling the launcher activity removes the app icon from the app launcher. Since the app is no longer displayed in the app launcher, though, spyware apps need a different way to launch themselves. We detail different ways to launch apps that are not present in the app launcher in Capability #6 below.

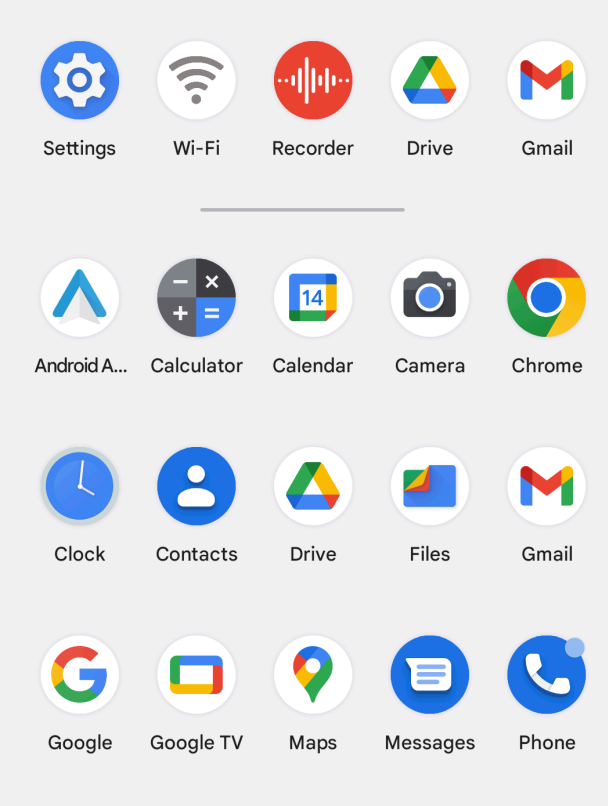

Figure 1: App launcher that displays app icons. The Spyhuman app installed itself as the innocuous "Wi-Fi" icon.

Since Android 10, an app's icon will appear in the app launcher even if the launcher activity is disabled at runtime. In reaction to this change, most spyware apps now use innocuous names and icons to disguise the spyware app, such as "Wi-Fi" (as in Figure 1), "SyncService" and "InternetService".[8] Overall, 11 apps hide their icons in this way.

Specifying no launcher activity lets spyware apps hide app icons even in the latest Android 12 and has not been noticed in any prior work. Android 10 effectively stopped apps from hiding their icons at runtime, it still leaves a loophole. An app that meets any of the following three requirements can still hide their icon from the launcher [53]: (1) it is a system app; (2) it does not request any permissions; or (3) the app does not have a launcher activity that is enabled by default (a launcher activity has an intent containing the `ACTION_MAIN` action and the `CATEGORY_LAUNCHER` category). Spyware apps abuse the third requirement and hide their app icons by specifying no launcher activity.[9]

[7] One exception is Japan, where muting the shutter sound when taking pictures or videos is not possible due to legal requirements [48].

[8] iKeyMonitor even offers the ability to build apps that have customized icons and names.

[9] We note that, for a short period of time (Android 10 builds r1 to r14 [54]), the third requirement was "the app does not have any components", which should stop the abuse described in this section (as spyware apps cannot operate without components).

We observe two spyware apps (Spy24 and Meuspy) that implement this technique: Spy24 only has an activity with an intent containing action `ACTION_MAIN` and `LEANBACK_LAUNCHER` (used by TV apps); Meuspy only has an activity with an intent containing the `ACTION_MAIN` action. Since both apps do not have an icon displayed in the app launcher, they cannot easily be started by a normal user. As a result, both Spy24 and Meuspy use a loader to start the app (as described in Appendix A.1).

We have disclosed the abuse of the `LEANBACK_LAUNCHER` intent on phone apps to Google, and Google determined that this abuse was not a security vulnerability.[10]

Capability #6: Launching a Hidden App. When spyware apps hide by excluding themselves from the launcher (Capability #5), they need an additional mechanism to launch.[11] One mechanism is to provide a way for the abuser to unhide the app icon on demand. We observed two ways apps unhide their app icon: by sending a command from the web portal or an SMS text from a phone. Apps can monitor SMS text messages by acquiring the `RECEIVE_SMS` permission, and can receive commands using third-party libraries such as Firebase [23]. Once launched, the abuser can send a similar command to hide the app icon again.

A second mechanism is to have apps launch themselves based on an external trigger, keeping the app icon hidden. This mechanism is generally achieved by receiving a predetermined signal from another app in the system via the Intents abstraction. We observed two main ways to do this signalling: (1) entering a hidden code via Android's dial pad, which is the predominant approach; and (2) entering a URL in the browser.

Apps that launch by entering a hidden code with Android's dial pad register a system broadcast receiver that listens for the `NEW_OUTGOING_CALL` intent. This intent is automatically broadcast by the system when there is an outgoing call, which in turn invokes the receiver registered by the app. Apps can monitor outgoing dialed numbers and launch themselves after a specific code (e.g., `*1234#`) is entered in the dial pad. Launching an app via a browser URL can be done by adding an intent filter for a specific URL (e.g., `www.myapp.com/launch`) in the manifest (e.g., as described in [55]). This intent filter specifies how to route an intent that has the matching URL.

Unique to Hoverwatch, besides implementing the mechanisms mentioned above, it also supports launching itself by entering a code through a widget: a miniature app view that can be embedded in other applications such as the home screen [56]. Upon installation, Hoverwatch creates a widget that has an `EditText` control that takes user input. If users enter the correct code, it launches itself.

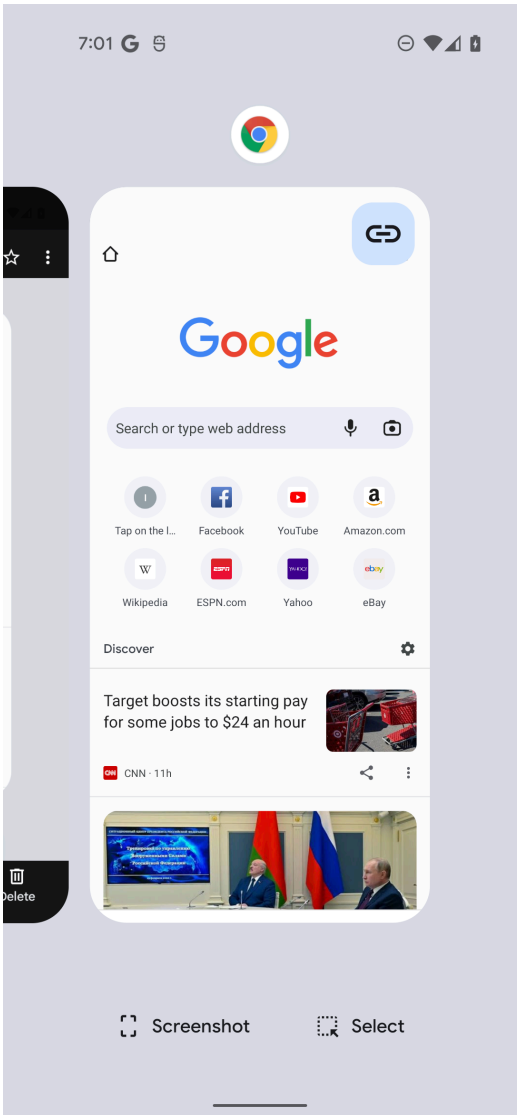

Figure 2: A screenshot of Recents screen showing that Chrome is recently accessed. However, a spyware app will not appear in the recent screen (if it chooses to hide from the recent screen), despite that one of its Activities is recently created and displayed.

Capability #7: Hiding From the Recents Screen. Android lists recently accessed activities and tasks in the Recents screen [57] (see Figure 2 for an example). Past literature [51, 58] has documented that the `android:excludeFromRecents` attribute can be abused to hide app activity, and this abuse is trivial to implement. It can be done via setting the `android:excludeFromRecents` attribute to true for relevant activities in their manifest [59] or programmatically adding the flag `FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS` [60] to an activity.

3.4.1 Mitigations. For apps that seek to hide their icons, our recommendation is that Android should enforce stricter requirements

Unfortunately, this requirement was changed to the current form since Android 10 build r15.

[10] They note that such "icon hiding" is against Google Play *policy*, but since none of the apps we studied are deployed via Google Play this restriction has no impact.

[11] We note that two apps, mSPY and Clevguard, do not provide ways to unhide icons even though they can hide icons. These two apps do not interact with users after hiding their icons upon installation, and configuring them post-installation is done remotely through the web portal. Additionally, while mSPY has implemented functions to unhide its icon via a remote command, it does not offer such an option on its web portal.

on what apps can hide icons (e.g., the three requirements from Android 10 build r1 to r14 mentioned in footnote 9). Most apps that run on Android phones should be required to have an icon. In the case of exploiting TV app features, while we understand that running apps with only `LEANBACK_LAUNCHER` activities on Android phones increases compatibility, such a feature leads to abuse as Spy24 has already demonstrated.

Launching apps by predetermined signal from another app is not only used for malicious purposes but also for benign purposes (e.g., dial pad can be used for testing and numerous benign apps use a browser link to open themselves). While browser links can be easily tracked by examining the manifest, currently users have no way to discover if apps can launch themselves with hidden codes. The difficulty is in part because hidden codes are used dynamically during runtime: the outgoing dialed number is sent to apps, and apps can freely decide what actions to perform based on the number received. This design makes it hard for the Android system to identify what hidden numbers are being tracked by each app. One possible mitigation is to allow users to review apps that can receive predetermined signals (e.g., a list of apps that listen for the `NEW_OUTGOING_CALL` intent or have intent filters for specific browser links, perhaps as part of the privacy dashboard).

One potential fix to stop apps from hiding from the Recents screen is to enforce having at least one activity per app in the Recents screen.

3.5 Persistence

This section describes the methods used by spyware apps to persist on the target device by obscuring the uninstallation process and creating "diehard" services (automatically restarting themselves after stops and reboots).

Capability #7: Obscuring the Uninstallation Process. One way to prevent users from stopping and uninstalling an app is by disabling the Force Stop and Uninstall buttons (see Figure 3 in the appendix). For Android versions below 7.1, these buttons can be disabled simply by registering the app with `Device Administrator` (DA) privileges (as detailed in prior work [61]). To enable these two buttons, the user would have to deactivate the device administrator privileges for the app. Since Android 7.1, while the Force Stop button is disabled for DA apps, the Uninstall button will remain enabled even if the app registers as DA. As a result, users can uninstall DA apps directly [61]. Overall, we observe 11 apps that register as DA.

Two apps, Cerberus and Mobile-tracker-free, directly interfere with the uninstallation process, a behavior often observed in Android malware [61, 62]. Cerberus employs a series of mechanisms to stop users from deactivating it as a DA app or uninstalling it. These include trying to lock the device by invoking the `lockNow` method of the `DevicePolicyManager` class and starting an activity that blocks users from clicking on any buttons. Mobile-tracker-free, on the other hand, tries to stop users from uninstalling it by starting an Activity that blocks the uninstallation screen and requests a password set by the abuser to proceed.

Capability #8: Creating Diehard Services. Spyware apps strive to always be executing on the target device so that they can collect as much information as possible. We focus on the "diehard" mechanisms that apps use to automatically restart themselves after being stopped by the Android system (e.g., due to low memory) or after device reboots. Echoing the diehard implementations discovered in prior work [58, 63], we observe two main ways used by spyware apps to create diehard services. We also note a third approach that appears to originate from the spyware ecosystem and be a byproduct of other capabilities.

System Broadcast	# of Apps
BOOT_COMPLETED	10
SMS_RECEIVED	9
NEW_OUTGOING_CALL	9
PHONE_STATE	6
ACL_DISCONNECTED	1
ACL_CONNECTED	1
LOCKED_BOOT_COMPLETED	1
WAP_PUSH_RECEIVED	1

Table 3: System broadcasts and the number of apps monitoring them.

Leveraging Scheduling Frameworks. Scheduling frameworks such as `JobScheduler` [64] and `AlarmManager` [65] enable apps to repeatedly restart. Apps can schedule to be restarted either when they are first started or when they are being terminated by the system. To schedule themselves to be restarted shortly after they are terminated, they override the `onDestory` function, which is called before the app is terminated by the system.

Monitoring System Broadcasts. Monitoring system broadcasts offers another way to wake up apps if they are not running already. Android sends broadcasts when various system events occur [66], and apps that monitor these system broadcasts will be woken up if they are not running already. Table 3 lists various systems broadcasts and the number of apps that monitor them.[12] The spyware apps in our study predominantly use the `BOOT_COMPLETED` broadcast. Monitoring `BOOT_COMPLETED` allows spyware apps to restart themselves after the device reboots: the Android system will send a `BOOT_COMPLETED` system broadcast upon reboot, and spyware apps that listen for this broadcast will automatically restart themselves. While `NEW_OUTGOING_CALL` and `SMS_RECEIVED` are also popular, we note that they serve dual purposes (`NEW_OUTGOING_CALL` can be used to launch a hidden app and `SMS_RECEIVED` can be used to monitor SMS messages).

Listening for Accessibility or Notification Events. Apps that register as an `AccessibilityService` or `NotificationListenerService` can also survive device reboots. However, unlike the two techniques described above, `AccessibilityService` is less reliable because it can be turned off by Android for battery saving reasons [68, 69].

3.5.1 Mitigations. Apps that seek to persist by registering as device administrator will no longer be successful as users can uninstall them on most devices running Android 7.1 or above. For apps that

[12] Only a small number system broadcasts can be used to wake up apps after the restrictions introduced in Android 8 [67]. We only consider these system broadcasts.

Spyware Apps	Eavesdropping Sensitive PII	Cross-account Request Forgery	Unauthenticated Access to Victim Data	Poor Data Retention Practices	Unauthenticated SMS Commands
mSPY	◒		Images		
Mobile-tracker-free			Streaming		●
Clevguard	◒		Images*		
HoverWatch			Audio*		
Flexispy	●		Images/Audio*		
Spyic			Images*	◒	
Spyhuman			Images/Audio		
TheTruthSpy	●	●	Images/Audio	●	
iKeyMonitor					
Cerberus			Audio		
Spy24			Streaming*		
Spapp			Images/Audio/Streaming	◒	●
Meuspy			Images/Audio	◒	
Highstermobile			Images		

Table 4: Systematization of commodity spyware vulnerabilities. Circles denote the severity level of the insecurity. ◒ indicates at least one instance of the insecurity; ● indicates all app functionality is insecure; * indicates URLs are temporary and expire. Shading added to improve readability.

seek to interfere with the uninstallation process by creating activities or locking the device, past literature on malware defense [62, 70] has suggested improving attack detection and introducing system level support for detecting and reacting when an app window is covered.

Prior work [58] has investigated various ways of creating diehard apps. According to the authors, the diehard mechanisms we observe are very resilient and can only be effectively stopped via a force stop. Additionally, the fact that many spyware apps register as device administrator creates another layer of complication — while apps registered as DA may be uninstalled directly after Android 7.1, the force stop option is disabled, making it difficult for users to stop the spyware apps (unless they uninstall it). However, even if apps prevent a force stop, users can still uninstall them. We also recommend adding a dashboard for monitoring apps that will automatically start themselves.[13]

4 SPYWARE USER DATA PROTECTION

In this section we assess how well spyware apps secure the user data they exfiltrate and store. For this assessment, in addition to the victim and the abuser, we consider a third actor: an external attacker who seeks to undermine the security of the spyware app. The attacker's goal is to exploit any integrity, authorization, and authentication issues with the spyware app to access victim data as a third party.

We start by describing our methodology for investigating the data protection practices of the spyware apps, and then discuss each of the vulnerabilities we uncovered (summarized in Table 4).

[13]We note that some Android phones (e.g., Xiaomi) have a built-in dashboard for managing apps that can automatically start themselves [71].

4.1 Methodology

For each app we analyzed, we obtained an account (providing payment information when necessary) and registered the app on a Pixel 2 XL Android phone running Android 10.

To identify apps sending data from the device to the backend servers via unencrypted connections we used `tcpdump`, listening on a Windows 10 machine interface configured as a mobile hotspot, to capture and inspect app traffic over the network. We configured the Android phone to connect to the PC and recorded all traffic exchanged with the app servers, without any proxy in between (avoiding any potential TLS handshake failures due to certificate pinning).

To study the interactions between the spyware web portal interface and the spyware backend servers, we used the open-source `MITMProxy` tool to decrypt HTTPS traffic. This configuration gave us access to authentication tokens (session cookies, secrets, etc.) for conducting forgery attacks on our test accounts. We were able to login to all portals without error (e.g., there were no issues with certificate pinning). We also analyzed the HTML content displayed via the spyware web portal interface, extracting the URLs linked to uploaded user media (images, audio, etc.). We then performed a sequence of experiments that tested and verified each insecurity listed in Table 4.

4.2 Results

Table 4 summarizes the threats we investigated and shows which apps are susceptible to them. We describe each of the threats in turn, summarizing its context, its associated threat model, any additional methods we employed, and the specific results we discovered for the vulnerable apps.

Insecurity #1: Eavesdropping Sensitive Personally Identifiable Information (PII). Some spyware apps transmit highly sensitive victim data,

such as photos, texts, and location, from the victim device to the spyware backend servers using unencrypted HTTP connections. A MITM attacker who eavesdrops on the same communication channel (e.g., same WiFi network) could collect all data and credentials sent unencrypted over the network. Furthermore, credentials leaked over the network enable an attacker to login to the abuser's account and gain access to all of the victim's data exfiltrated by the spyware.

Threat Model. We assume that the attacker can eavesdrop on all messages sent by the mobile device infected with spyware (e.g., the attacker uses the same WiFi network as the victim, and the WiFi network is not using link-layer encryption).

Experimental Setup. We filtered the captured traffic to observe network activity from each app over insecure channels like HTTP, and used a combination of search and manual inspection to analyze the recorded traffic for sensitive data such as leaked credentials (user names, email addresses, passwords), text messages, etc.

Results. Four spyware apps in our study leak at least some of their data using vulnerable communication channels. TheTruthSpy submits all of its data over HTTP, and it leaks the abuser's credentials for TheTruthSpy servers (abuser email address and password) in the authentication request during app setup. mSPY leaks only the upload path for its images in an HTTP response back to the victim's device when the image upload succeeds. Clevguard uploads its images over HTTP, making it is possible for an attacker to reconstruct the image from the payload. Flexispy uses HTTP for all of its communication, but it implements a custom encryption protocol for the data it sends. However, a previously discovered flaw in Flexispy's encryption makes it possible to intercept personal data sent over the network [72].

Insecurity #2: Cross-account Request Forgery. Knowing a particular user identifier, or a file path that provides access to images, audio, or video on the server, an attacker can use valid cookies from one spyware account to access data or perform actions in the accounts of other spyware users.

Threat Model. The attacker can register a spyware account and log into it. Using a MITM traffic inspector (similar to our proxy described in Section 4.1), an attacker can intercept and replay authenticated requests from their account. We assume the attacker knows the ID of the targeted user's account — either through enumeration, or from the ID being leaked over the network or in a public data dump [5–7, 9, 10].

Experimental Setup. We used two accounts for each app, one for the attacker and the other as the targeted spyware user. We then recorded and replayed authenticated requests from the attacker's account to the targeted user's account, substituting the ID of the targeted user's account in the requests without changing authorization tokens provided by the attacker's account (e.g., session IDs, API keys). We only replayed requests to our test accounts, and made no attempt to access content belonging to any other account.

Results. Only one of the apps in our study is vulnerable to cross-account request forgery. With TheTruthSpy, it was possible to access all data (messages, contacts, locations, images, etc.) in the targeted account using a cookie provided by the attacker's account, and simply swapping the attacker's device ID with the targeted account's device ID in requests to TheTruthSpy server. Furthermore, when we registered multiple test accounts in succession, we noticed that the device ID that TheTruthSpy assigns to a new device is a six-digit integer which increases incrementally when new devices are registered. The combination of insecure access controls across accounts and predictable device IDs makes it possible for an attacker to retrieve data from other accounts without needing to identify the device ID beforehand.

Insecurity #3: Unauthenticated Access to Victim Data. Many of the spyware apps use different backend infrastructure to store and deliver media data that is distinct from the servers that abusers use to access the app dashboard (e.g., abusers login to the dashboard via the website domain in Table 1, but the app may use a CDN to deliver images and videos). This separate infrastructure includes cloud storage (e.g., AWS S3 buckets), content distribution networks, and other shared hosting services, all of which are presumably cheaper to use for serving media data. Some of the spyware apps are less careful about protecting the data that they store on this hosting infrastructure, often allowing unauthenticated access to the URLs they generate to stored media data. Further, we observed user data stored in predictable URLs that make it possible to access data across different accounts, protected solely through security by obscurity and vulnerable to enumeration. Table 4 lists the kinds of data leaked in public URLs for the apps studied. Failure to authenticate access to user media (images, audio, etc.) using mechanisms like cookies is a common example of this category of vulnerability.

Threat Model. We assume that the attacker has knowledge of the media upload path for the app based on studying media paths revealed using accounts owned by the attacker. For apps that use the device ID in URL path components, we also assume the attacker knows the targeted device ID.

Experimental Setup. To discover sensitive data leakage, we focused primarily on URLs used to store media artifacts (images, audio, video, etc.). We extracted these URLs by examining the browser-generated HTML content when accessing the dashboard in the spyware account. We only attempted to access media upload paths from our test accounts, and made no attempt to access content belonging to any other account.

Results. Six of the apps in our study store their data in public URLs accessible without authenticated access. We disregarded apps which store data in URLs that, although public, expire after a short duration (e.g., Spyic's links to images expire after 1,800 seconds).

Highstermobile stores its images with a URL scheme as a concatenation of the image upload timestamp (UNIX timestamp with seconds precision) and a double digit random number, with no other per-device ID in the path (e.g., https://domain.com/path/photo_<timestamp><00-99>.png). Since an attacker could plausibly iterate over a short range of timestamps and random digits, this naming scheme makes it straightforward for an attacker to gain unauthorized access to media stored on the server for other accounts.

Cerberus and Spyhuman use public URLs that are a combination of device ID and Unix timestamp (e.g., https://domain.com/<DeviceID>-<timestamp>.ext). While both of them require the attacker to know the device ID before hand, an attacker could for instance obtain the device ID from data dumps made public in breach attacks [5–7, 9, 10] or with physical access to the device. Cerberus stores audio insecurely using the device IMEI as the device

ID. Spyhuman stores both its images and audio insecurely using the Android serial number as its device ID. With both apps, an attacker knowing the device ID could plausibly iterate over a short range of timestamps to retrieve data stored on the server.

Three other apps store data on servers using public URLs that rely on security through obscurity, but they generate URLs that are neither enumerable or predictable. For instance, Spy24 allows WebRTC remote streaming through a public URL based on a request ID generated by the app, Spapp requires two alphanumeric keys to access image and audio files on the server, and Meuspy generates unguessable URLs to store images and audio. Although certainly not good security practice, the risk associated with unauthenticated access is lower than for the other apps we discussed.

Insecurity #4: Poor Data Retention Practices. Some spyware apps have poor data retention practices that prevent victim data from being deleted from the spyware servers. These data retention issues pose serious security and privacy risks [73]. A data breach could expose residual victim PII which has persisted even if, for instance, the abuser has deleted their account.

Threat Model. Data uploaded to the spyware servers and never deleted poses a long-term risk to being made public in data dumps associated with breach attacks.

Experimental Setup. To discover poor data retention practices, we analyzed network traffic generated after the app license expired, but with the victim device still logged in. We also tested access to media on deleted accounts using their public URLs up to seven days after deletion of the account.

Results. Four of the apps in our study demonstrate poor data retention practices.

Consistent with its other data vulnerabilities, TheTruthSpy also has data retention issues. It continues to send data from the victim's phone (e.g., text messages, images, keylogger activity) to its servers after the app license expires. Further, it persists some data after the abuser deletes their spyware portal account and the data associated with it. In particular, images uploaded from the victim device persist after account deletion, and the images remained accessible through the public URLs used to access them (Section 4.2).

Spyic has a data retention issue resulting from its trial usage model. When installed for trial use, Spyic uploads data from the victim device to the server. But it only allows access to its portal after the abuser has purchased a subscription. If the abuser decides not to purchase the spyware, victim data persists indefinitely on Spyic's servers (presumably in anticipation of the spyware user eventually deciding to pay for a license).

Lastly, both Spapp and Meuspy persist data from an abuser's account after deletion. Images in particular continue to be accessible through the public URLs used to access them.

Insecurity #5: Unauthenticated SMS Commands. Four of the spyware apps we studied accept commands in the form of SMS messages. This capability enables the abuser to control the spyware app on the victim device even if it is disconnected from the Internet, or as a secondary form of control to the web portal accessible via the abuser's account.

Threat Model. We assume that the attacker knows the phone number of the target device (or can enumerate it). We also assume that the attacker knows the list of commands available by referencing online app documentation for SMS commands, such as [74, 75].

Experimental Setup. For each app, we systematically tested their SMS commands on our test device, focusing in particular on their method of authenticating commands.

Results. Two of the four apps that accept SMS commands require a strong password or license key to authenticate SMS commands, and so we disregarded them from further analysis. The remaining two apps fail to check if the text message is from an authorized sender, and execute the commands regardless.

Spapp executes highly-sensitive SMS commands, such as remotely wiping the victim's phone, effectively unauthenticated. During app setup, the app generates a random two-digit number that serves as a passcode, and only allows SMS commands that include the correct passcode to execute on the device. However, this simple passcode provides little security since a methodical attacker can send redundant commands enumerating all possible passcodes.

Mobile-tracker-free authenticates its SMS commands, but uses a default constant as its password. Unless the abuser explicitly changes this SMS password, an attacker can successfully send commands using the default. However, the set of commands it supports via SMS is more restricted than Spapp (e.g., the command `MobileTrackerFreeSMS--getlocation` returns device location information in an SMS response).

4.3 Disclosure

We disclosed the findings in this section to all the affected vendors on June 14th, 2022. No vendor has replied to our disclosures as of the date of publication (three months after our disclosure).

5 RELATED WORK

There exists a rich literature from both academia and industry that examines various aspects of spyware apps (e.g., their usage in the context of intimate partner violence).

Most related to our work, several prior studies have examined the technical capabilities of spyware apps, including both industry reports [41, 72, 76–85] and academic papers [86–91]. However, many of these efforts focus on documenting the functionalities supported by the spyware apps and do not shed light on the implementation used to achieve different functionalities (mostly because they focus on other facets instead of the technical implementation challenges). The ones [41, 72, 79, 80, 82, 83, 86] that do study the implementation, either examined only one or two apps or a small subset of the mechanisms employed. Our work builds on these studies by systematically and comprehensively analyzing the underlying technical methods that apps employ to acquire different spying capabilities.

Also related, but orthogonal, is work focused on identifying and detecting spyware apps, both industrial reports listing such apps [18, 19, 92] and academic efforts to characterize and build detection algorithms for them [3, 89, 93–100]. Yet another related body of work examines spyware apps' presence in different contexts such as intimate partner violence and cyberstalking [4, 101–112]. We believe our findings, particularly characterizing the data access mechanisms used by spyware, will be of use to those implementing detectors, but detection is not itself a goal of our work.

Outside the context of spyware, another related research domain has focused on how various kinds of malware (including spyware) can abuse Android APIs to achieve abusive functionality. In particular, several papers have also identified abuse of the Android Accessibility APIs, starting with Kraunelis et al. [35]. Following this line of work, Fratantonio et al. [34], Kalysch et al. [38], Diao et al. [37], and Naseri et al. [40] have documented how Accessibility can be abused in various contexts. While several of these papers suggest potential fixes, Huang et al. [39] is the first to describe a comprehensive framework for mitigating misuse in the accessibility API. Others have explored other forms of API abuse, including Audio and Video APIs [49, 113], screenshot API [114], device administration APIs [61], WebView-related APIs [115–118], the use of overlays in malware [24], and mechanisms for app hiding, discovery [51, 119] and persistence [58]. Our work builds on all of these efforts, but rather than exploring these issues abstractly, focuses specifically on how they manifest in consumer mobile spyware in the wild. Our detailed analysis not only confirmed that the consumer spyware sector exploits similar techniques documented in broader mobile malware, but also uncovered two new forms of API abuse (invisible camera access and hiding app icons) that appears to have originated from within the spyware ecosystem.

Finally, spyware companies have a long history of poor security hygiene. Numerous media reports describe data breaches at various spyware companies, including Spyhuman [5], TheTruthSpy [6], mSPY [7, 8], Cerberus [9], Flexispy [10], Mobistealth [11], Spyfone [12], Retina-X [13, 14], among others. These breaches have exposed hundreds of thousands (if not millions) of users' sensitive personal information (e.g., location, videos, etc.) to the broad public. Our work is responsive to these events and seeks to explore the nature of the security protections provided by spyware vendors and the extent to which these breaches have led to improved practices. While recent, contemporaneous report from ESET [18] also investigated similar issues, our work is distinct in analyzing the security of each app from the context of protecting user data and presents a detailed, documented, and reproducible methodology.

6 CONCLUSION

Consumer mobile spyware persists because it exists in a gray area: not clearly legal, but not canonically illegal; not allowed in the app store, but broadly available via side loading; not supported by APIs but able to achieve its ends through manipulation and trickery; repeatedly breached, but able to maintain market power because those injured are not its customers.

For example, the use of such software to monitor arbitrary individuals without consent is clearly illegal — both due to violations of the Computer Fraud and Abuse Act (18 USC 1030) and provisions of the Wiretap Act (18 USC 2511). However, contemporary mobile spyware companies argue that they do not support or encourage such uses. Indeed, since the Department of Justice brought a criminal indictment against the makers of StealthGenie [120] in 2014, spyware vendors have generally restricted their public marketing to focus on the monitoring of minor children (whose consent is abdicated to their guardians) or the monitoring of employees (such monitoring can be viewed as consensual when the equipment is owned by the employer and employees are clearly informed about the policies around monitoring). However, this shift in "official" marketing has done little to undermine the large market for using this software illegally and a broad array of sites and forums provide detailed direction on how to use such apps to covertly monitor a spouse or partner.

Similarly, while curated app stores, such as Google Play Store, now disallow such apps from being sold, Android's default support for sideloading makes this limitation only a minor obstacle for someone seeking to surreptitiously install spyware on a phone.

Moreover, the fact that spyware abusers are able to obtain physical access to a device (at least temporarily), renders Android's finer-grained permissions checks ineffectual as well. The one-time "consent" provided by the spyware installer provides largely unfettered capabilities that the true user may never be aware of. The Accessibility API offers a particularly large consent loophole, as its intended function necessitates almost complete mediation of I/O activities. Moreover, even when the API itself has been changed to restrict certain capabilities we have repeatedly found spyware authors creatively abusing APIs or their implementations to gain capabilities that were not meant to be available to third-party apps (e.g., the range of mechanisms described in Section 3.3 for covertly performing audio recording in spite of multiple OS changes intended to prevent such abilities). We uncover Android's incomplete threat model with our discovery of their unwillingness to fix what we consider to be a vulnerability in their API that allows spyware apps to hide their icon.

The privacy deficiencies we uncover in Section 4, on the other hand, demonstrated the unfortunate truth about consumer mobile spyware apps: that they prioritize covert collection over protecting user data. As an example, Spapp shows signs of significant developer effort: it implements most of the technical collection capabilities we have described and carefully obfuscates its code to hinder reverse engineering efforts. However, the same app places little investment in protecting the data it has collected, incorrectly handling data retention after deletion and executing highly sensitive SMS commands without authentication. Sadly, this situation is far from the exception — and the range of past data breaches are testament to this asymmetry. Moreover, because it is victims who suffer here and not spyware customers, there are no market forces that will correct this state of affairs.

All of these challenges highlight the need for a more creative, diverse and comprehensive set of interventions from industry, government and the research community. While technical defenses can be part of the solution (and particularly OS improvements that make users aware of their *current* exposure, like the new privacy dashboard in Android 12), consumer spyware's persistence and growth suggests that a broader range of measures including payment interventions [121], regulatory crackdowns (e.g., FTC recently banned SpyFone from operating [122]) and further law enforcement action may also be necessary to prevent surveillance from becoming a consumer commodity.

ACKNOWLEDGMENTS

We thank our shepherd Alastair Beresford and anonymous reviewers for their insightful and constructive suggestions and feedback. We thank Cindy Moore and Jennifer Folkestad for their operational

support. We thank Thomas Ristenpart and Nicola Dell for helping shape this project in its early stage. We thank Nishant Bhaskar and Ariana Mirian for supplying test phones. We thank Brad Chen for collecting and providing feedback from Google. We thank Earlence Fernandes for his feedback on this paper. We thank Weijia He for her feedback on formatting this paper. Funding for this work was provided in part by National Science Foundation grant CNS-1916126, the UCSD CSE Postdoctoral Fellows program, the Irwin Mark and Joan Klein Jacobs Chair in Information and Computer Science, and operational support from the UCSD Center for Networked Systems.

REFERENCES

[1] N. Labs. (2022, 02) A year after lockdown: Stalkerware on the rise. [Online]. Available: https://www.nortonlifelock.com/blogs/norton-labs/stalkerware-rise
[2] Avast. (2022, 02) Use of stalkerware and spyware apps increase by 93% since lockdown began in the uk. [Online]. Available: https://press.avast.com/use-of-stalkerware-and-spyware-apps-increase-by-93-since-lockdown-began-in-the-uk
[3] R. Chatterjee, P. Doerfler, H. Orgad, S. Havron, J. Palmer, D. Freed, K. Levy, N. Dell, D. McCoy, and T. Ristenpart, "The Spyware Used in Intimate Partner Violence," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy*, 2018, pp. 441–458.
[4] D. Woodlock, "The Abuse of Technology in Domestic Violence and Stalking," *Violence Against Women*, vol. 23, no. 5, pp. 584–602, 2017.
[5] J. Cox. (2022, 02) Hacker steals customers' text messages from android spyware company. [Online]. Available: https://www.vice.com/en/article/qvm44m/hacker-steals-text-messages-android-spyware-company-spyhuman
[6] Waqas. (2022, 02) Company that sells spyware to domestic abusers hacked. [Online]. Available: https://www.hackread.com/company-that-sells-spyware-to-domestic-abusers-hacked/
[7] B. Krebs. (2022, 02) mspy breach krebs on security. [Online]. Available: https://krebsonsecurity.com/tag/mspy-breach/
[8] Cyber Insurance. (2022, 02) mspy - cyberinsurance.com. [Online]. Available: https://www.cyberinsurance.com/breaches/mspy/
[9] Rithvik. (2022, 02) Cerberus acknowledges data breach, states some usernames and encrypted passwords stolen. [Online]. Available: https://www.droid-life.com/2014/03/26/cerberus-data-breach/
[10] L. Franceschi-Bicchierai. (2022, 02) Stalkerware company flexispy calls catastrophic hack 'just some false news'. [Online]. Available: https://www.vice.com/en/article/xyjwpw/flexispy-calls-catastrophic-hack-just-some-false-news
[11] J. Cox. (2022, 02) Hacker strikes 'stalkerware' companies, stealing alleged texts and gps locations of customers. [Online]. Available: https://www.vice.com/en/article/7x77ex/hacker-strikes-stalkerware-companies-stealing-alleged-texts-and-gps-locations-of-customers
[12] C. Osborne. (2022, 02) Spyware firm spyfone leaves customer data, recordings exposed online. [Online]. Available: https://www.zdnet.com/article/spyware-firm-spyfone-leaves-customer-data-recordings-exposed-online/
[13] Z. Zorz. (2022, 02) Retina-x admits they have suffered a data breach - help net security. [Online]. Available: https://www.helpnetsecurity.com/2017/05/02/retina-x-data-breach/
[14] L. Vaas. (2022, 02) Hacker claims spyware maker retina-x has been breached, again. [Online]. Available: https://nakedsecurity.sophos.com/2018/02/23/hacker-claims-spyware-maker-retina-x-has-been-breached-again/
[15] Wikipedia. (2022, 08) Insecure direct object reference. [Online]. Available: https://en.wikipedia.org/wiki/Insecure_direct_object_reference
[16] Common Weakness Enumeration. (2022, 08) Cwe - cwe-813: Owasp top ten 2010 category a4. [Online]. Available: https://cwe.mitre.org/data/definitions/813.html
[17] Z. Whittaker. (2022, 02) How to jailbreak your iphone or ipod touch. [Online]. Available: https://www.digitaltrends.com/mobile/how-to-jailbreak-your-iphone/
[18] L. Stefanko. (2021, 05) Android stalkerware vulnerabilities. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2021/05/eset_android_stalkerware.pdf
[19] Te-k. (2022, 01) Te-k/stalkerware-indicators: Indicators of stalkerware apps. [Online]. Available: https://github.com/Te-k/stalkerware-indicators
[20] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," *arXiv preprint arXiv:1806.01156*, 2018.
[21] Apktool. (2022, 01) Apktool - a tool for reverse engineering 3rd party, closed, binary android apps. [Online]. Available: https://ibotpeaches.github.io/Apktool/
[22] Skylot. (2022, 01) skylot/jadx: Dex to java decompiler. [Online]. Available: https://github.com/skylot/jadx
[23] Google. (2022, 02) Firebase messaging service. [Online]. Available: https://firebase.google.com/docs/reference/android/com/google/firebase/messaging/FirebaseMessagingService
[24] Y. Yan, Z. Li, Q. A. Chen, C. Wilson, T. Xu, E. Zhai, Y. Li, and Y. Liu, "Understanding and Detecting Overlay-Based Android Malware at Market Scales," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019, pp. 168–179.
[25] Google. (2021, 08) Camera capture sessions and requests. [Online]. Available: https://developer.android.com/training/camera2/capture-sessions-requests
[26] ——. (2022, 02) Surfacetexture. [Online]. Available: https://developer.android.com/reference/android/graphics/SurfaceTexture
[27] ——. (2022, 02) Webview. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView
[28] ——. (2021, 08) Mediarecorder.audiosource. [Online]. Available: https://developer.android.com/reference/android/media/MediaRecorder.AudioSource
[29] ——. (2021, 08) Voice_call audio source requires android.permission.capture_audio_output [37094464] - visible to public - issue tracker. [Online]. Available: https://issuetracker.google.com/issues/37094464
[30] V. Degtyarev. (2021, 08) Viktordegtyarev/callreclib: Call recorder fix for android 7 and android 6. [Online]. Available: https://github.com/ViktorDegtyarev/CallRecLib
[31] Bitbucket. (2021, 08) copluk / acr / issues / #2418 - [kb] android 9 (p) call recording issues. [Online]. Available: https://bitbucket.org/copluk/acr/issues/2418/kb-android-9-p-call-recording-issues
[32] Google. (2021, 08) services/audioflinger/audioflinger.cpp - platform/frameworks/av - git at google. [Online]. Available: https://android.googlesource.com/platform/frameworks/av/+/android-9.0.0_r30/services/audioflinger/AudioFlinger.cpp
[33] ——. (2021, 08) Sharing audio input. [Online]. Available: https://developer.android.com/guide/topics/media/sharing-audio-input#accessibility_service_ordinary_app
[34] Y. Fratantonio, C. Qian, S. P. Chung, and W. Lee, "Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, 2017, pp. 1041–1057.
[35] J. Kraunelis, Y. Chen, Z. Ling, X. Fu, and W. Zhao, "On Malware Leveraging the Android Accessibility Framework," in *Proceedings of the 2013 International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, 2013, pp. 512–523.
[36] Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee, "A11y Attacks: Exploiting Accessibility in Operating Systems," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 103–115.
[37] W. Diao, Y. Zhang, L. Zhang, Z. Li, F. Xu, X. Pan, X. Liu, J. Weng, K. Zhang, and X. Wang, "Kindness Is a Risky Business: On the Usage of the Accessibility APIs in Android," in *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses*, 2019, pp. 261–275.
[38] A. Kalysch, D. Bove, and T. Müller, "How Android's UI Security Is Undermined by Accessibility," in *Proceedings of the 2nd Reversing and Offensive-oriented Trends Symposium*, 2018, pp. 1–10.
[39] J. Huang, M. Backes, and S. Bugiel, "A11y and Privacy Don't Have to Be Mutually Exclusive: Constraining Accessibility Service Misuse on Android," in *Proceedings of the 30th USENIX Security Symposium*, 2021.
[40] M. Naseri, N. P. Borges, A. Zeller, and R. Rouvoy, "AccessiLeaks: Investigating Privacy Leaks Exposed by the Android Accessibility Service," *Proceedings of Privacy Enhancing Technologies*, vol. 2019, pp. 291–305, 2019.
[41] R. Gibson. (2019, 10) Countering tech abuse together. [Online]. Available: https://www.virusbulletin.com/uploads/pdf/conference_slides/2019/VB2019-ZakorzhevskyG.pdf
[42] Read Unread. (2022, 08) Read unread: unseen hide and read, last seenonline. [Online]. Available: https://play.google.com/store/apps/details?id=com.read.unread.last.seen.unseen.hidden.chat
[43] Stackoverflow. (2021, 10) android - how to use mediaprojection to capture screen in a service? - stack overflow. [Online]. Available: https://stackoverflow.com/questions/51182557/how-to-use-mediaprojection-to-capture-screen-in-a-service
[44] Google. (2021, 12) Restrictions on starting activities from the background. [Online]. Available: https://developer.android.com/guide/components/activities/background-starts
[45] ——. (2021, 12) Accessibilityservice. [Online]. Available: https://developer.android.com/reference/android/accessibilityservice/AccessibilityService#takeScreenshot(int,%20java.util.concurrent.Executor,%20android.accessibilityservice.AccessibilityService.TakeScreenshotCallback)
[46] ——. (2021, 11) Privacy changes in android 10. [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes
[47] Joao Apps. (2022, 08) Solved - clipboard monitor/listener no longer works on android 10. [Online]. Available: https://forum.joaoapps.com/index.php?threads/clipboard-monitor-listener-no-longer-works-on-android-10.49808/
[48] Google. (2021, 08) How can i turn off camera shutter sound - google pixel community. [Online]. Available: https://support.google.com/pixelphone/thread/

69083830/how-can-i-turn-off-camera-shutter-sound?hl=en
[49] G. Petracca, Y. Sun, T. Jaeger, and A. Atamli, "Audroid: Preventing Attacks on Audio Channels in Mobile Devices," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 181–190.
[50] Stackoverflow. (2021, 11) what are the uses of main, default and launcher in manifest file in android - stack overflow. [Online]. Available: https://stackoverflow.com/questions/9721030/what-are-the-uses-of-main-default-and-launcher-in-manifest-file-in-android
[51] Z. Shan, I. Neamtiu, and R. Samuel, "Self-Hiding Behavior in Android Apps: Detection and Characterization," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 728–739.
[52] Google. (2022, 02) Intent. [Online]. Available: https://developer.android.com/reference/android/content/Intent#CATEGORY_LAUNCHER
[53] ——. (2021, 08) Launcherapps. [Online]. Available: https://developer.android.com/reference/android/content/pm/LauncherApps#getActivityList(java.lang.String,%20android.os.UserHandle)
[54] LauncherAppsService. (2022, 08) Launcherappsservice.java - android code search. [Online]. Available: https://cs.android.com/android/platform/superproject/+/android-10.0.0_r1:frameworks/base/services/core/java/com/android/server/pm/LauncherAppsService.java;l=439
[55] Google. (2022, 02) Create deep links to app content. [Online]. Available: https://developer.android.com/training/app-links/deep-linking
[56] ——. (2022, 02) App widgets overview. [Online]. Available: https://developer.android.com/guide/topics/appwidgets/overview
[57] ——. (2021, 11) Recents screen. [Online]. Available: https://developer.android.com/guide/components/activities/recents
[58] H. Zhou, H. Wang, Y. Zhou, X. Luo, Y. Tang, L. Xue, and T. Wang, "Demystifying Diehard Android Apps," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 187–198.
[59] Google. (2021, 11) <activity>. [Online]. Available: https://developer.android.com/guide/topics/manifest/activity-element#exclude
[60] ——. (2022, 02) Intent. [Online]. Available: https://developer.android.com/reference/android/content/Intent#FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS
[61] Z. Shan, R. Samuel, and I. Neamtiu, "Device Administrator Use and Abuse in Android: Detection and Characterization," in *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.
[62] A. AlJarrah and M. Shehab, "Maintaining User Interface Integrity on Android," in *Proceedings of the 40th Annual Computer Software and Applications Conference*, vol. 1. IEEE, 2016, pp. 449–458.
[63] Y. Shao, R. Wang, X. Chen, A. M. Azab, and Z. M. Mao, "A Lightweight Framework for Fine-Grained Lifecycle Control of Android Applications," in *Proceedings of the 2019 EuroSys Conference*, 2019, pp. 1–14.
[64] Google. (2021, 08) Jobscheduler. [Online]. Available: https://developer.android.com/reference/android/app/job/JobScheduler
[65] ——. (2021, 08) Alarmmanager. [Online]. Available: https://developer.android.com/reference/android/app/AlarmManager
[66] ——. (2022, 06) Broadcasts overview. [Online]. Available: https://developer.android.com/guide/components/broadcasts
[67] ——. (2022, 06) Implicit broadcast exceptions. [Online]. Available: https://developer.android.com/guide/components/broadcast-exceptions
[68] Accountable2you. (2022, 08) Android accessibility keeps turning off accountable2you - accountable2you support. [Online]. Available: https://support.accountable2you.com/article/754-android-accessibility-keeps-turning-off-accountable2you#:~:text=If%20you%20notice%20that%20Accountable2You,to%20customize%20these%20battery%20settings.
[69] Stackexchange. (2022, 08) Accessibility services gets disabled automatically - android enthusiasts stack exchange. [Online]. Available: https://android.stackexchange.com/questions/137195/accessibility-services-gets-disabled-automatically
[70] E. Fernandes, Q. A. Chen, J. Paupore, G. Essl, J. A. Halderman, Z. M. Mao, and A. Prakash, "Android Ui Deception Revisited: Attacks and Defenses," in *Proceedings of the 2016 International Conference on Financial Cryptography and Data Security*, 2016, pp. 41–59.
[71] P. Mitchell. (2021, 04) How to disable auto-start apps on android - techcult. [Online]. Available: https://techcult.com/how-to-disable-auto-start-apps-on-android/
[72] A. Langton. (2019, 12) Stalking stalkerware: A deep dive into flexispy. [Online]. Available: https://blogs.juniper.net/en-us/threat-research/stalking-stalkerware-a-deep-dive-into-flexispy-2
[73] P. Santhanam, H. Dang, Z. Shan, and I. Neamtiu, "Scraping Sticky Leftovers: App User Information Left on Servers After Account Deletion," in *Proceedings of the 2022 IEEE Symposium on Security and Privacy*, 2022, pp. 2145–2160.
[74] S. Monitoring. (2022, 02) Available sms commands for spapp. [Online]. Available: https://www.spappmonitoring.com/news/display/live_control
[75] Flexispy. (2022, 02) Remote commands for flexispy. [Online]. Available: https://portal.flexispy.com/help/en/misc/sms-commands.html
[76] J. Dalman. (2015, 07) Commercial spyware — detecting the undetectable. [Online]. Available: https://www.blackhat.com/docs/us-15/materials/us-15-Dalman-Commercial-Spyware-Detecting-The-Undetectable.pdf
[77] M. Robinson and C. Taylor. (2020, 02) Spy vs spy: Spying on mobile device spyware. [Online]. Available: https://media.defcon.org/DEF%20CON%2020/DEF%20CON%2020%20presentations/DEF%20CON%2020%20-%20Robinson-Spy-vs-Spy.pdf
[78] Zscaler. (2019, 11) A new wave of stalkerware apps. [Online]. Available: https://www.zscaler.com/blogs/security-research/new-wave-stalkerware-apps
[79] ——. (2018, 10) Why you shouldn't trust "safe" spying apps. [Online]. Available: https://www.zscaler.com/blogs/security-research/why-you-shouldnt-trust-safe-spying-apps
[80] M. Grassi. (2014, 10) Reverse engineering of a commercial spyware for ios and android - speaker deck. [Online]. Available: https://speakerdeck.com/marcograss/reverse-engineering-of-a-commercial-spyware-for-ios-and-android
[81] Cyberarch Admin. (2021, 11) Your infosec s.w.a.t team. [Online]. Available: https://cyberarch.eu/our-blog/pegasus-spyware-analysis/
[82] Cyber Merchants of Death. (2017, 04) Flexspy application analysis. [Online]. Available: http://www.cybermerchantsofdeath.com/blog/2017/04/22/FlexiSpy.html
[83] Diskurse. (2022, 01) diskurse/android-stalkerware: Various analysis of android stalkerware. [Online]. Available: https://github.com/diskurse/android-stalkerware
[84] Zscaler. (2022, 05) Spyware presence in enterprise networks blog. [Online]. Available: https://www.zscaler.com/blogs/security-research/spyware-presence-enterprise-networks
[85] S. Sidor. (2022, 05) Android: apps can take photos with your phone without you knowing. - mobile security - romanian security team. [Online]. Available: https://rstforums.com/forum/topic/79016-android-apps-can-take-photos-with-your-phone-without-you-knowing/
[86] C. Parsons, A. Molnar, J. Dalek, J. Knockel, M. Kenyon, B. Haselton, C. Khoo, and R. Deibert, "The Predator in Your Pocket: A Multidisciplinary Assessment of the Stalkerware Application Industry," 2019.
[87] D. Harkin and A. Molnar, "The Consumer Spyware Industry: An Australian-Based Analysis of the Threats of Consumer Spyware," *Australian Communications Consumer Action Network*, 2019.
[88] D. Harkin, A. Molnar, and E. Vowles, "The Commodification of Mobile Phone Surveillance: An Analysis of the Consumer Spyware Industry," *Crime, Media, Culture*, vol. 16, no. 1, pp. 33–60, 2020.
[89] F. Pierazzi, G. Mezzour, Q. Han, M. Colajanni, and V. Subrahmanian, "A Data-Driven Characterization of Modern Android Spyware," *ACM Transactions on Management Information Systems*, vol. 11, no. 1, pp. 1–38, 2020.
[90] Á. Feal, P. Calciati, N. Vallina-Rodriguez, C. Troncoso, and A. Gorla, "Angel or Devil? A Privacy Study of Mobile Parental Control Apps," *Proceedings of Privacy Enhancing Technologies*, vol. 2020, no. 2, pp. 314–335, 2020.
[91] D. Harkin and A. Molnar, "Operating-System Design and Its Implications for Victims of Family Violence: The Comparative Threat of Smart Phone Spyware for Android Versus iPhone Users," *Violence Against Women*, vol. 27, no. 6-7, pp. 851–875, 2021.
[92] Ch33r10. (2022, 02) ch33r10/stalkerware. [Online]. Available: https://github.com/ch33r10/Stalkerware
[93] M. Almansoori, A. Gallardo, J. Poveda, A. Ahmed, and R. Chatterjee, "A Global Survey of Android Dual-Use Applications Used in Intimate Partner Surveillance," *Proceedings of Privacy Enhancing Technologies*, vol. 2022, pp. 120–139, 2022.
[94] Y. Han, K. A. Roundy, and A. Tamersoy, "Towards Stalkerware Detection With Precise Warnings," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 957–969.
[95] S. Saroiu, S. D. Gribble, and H. M. Levy, "Measurement and Analysis of Spyware in a University Environment." in *Proceedings of the 2004 USENIX Conference on Networked Systems Design and Implementation*, 2004, pp. 141–153.
[96] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song, "Dynamic Spyware Analysis," in *Proceedings of the 2007 USENIX Annual Technical Conference*, 2007.
[97] K. A. Roundy, P. B. Mendelberg, N. Dell, D. McCoy, D. Nissani, T. Ristenpart, and A. Tamersoy, "The Many Kinds of Creepware Used for Interpersonal Attacks," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy*, 2020, pp. 626–643.
[98] H. Wang, S. Jha, and V. Ganapathy, "NetSpy: Automatic Generation of Spyware Signatures for NIDS," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, 2006, pp. 99–108.
[99] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, "A Crawler-Based Study of Spyware in the Web." in *Proceedings of the 2006 Network and Distributed System Security Symposium*, 2006.
[100] A. Randall, E. Liu, G. Akiwate, R. Padmanabhan, G. M. Voelker, S. Savage, and A. Schulman, "Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers," in *Proceedings of the 2020 ACM Internet Measurement Conference*, 2020, pp. 50–64.

[101] S. Havron, D. Freed, R. Chatterjee, D. McCoy, N. Dell, and T. Ristenpart, "Clinical Computer Security for Victims of Intimate Partner Violence," in *Proceedings of the 28th USENIX Security Symposium*, 2019, pp. 105–122.
[102] D. Freed, S. Havron, E. Tseng, A. Gallardo, R. Chatterjee, T. Ristenpart, and N. Dell, ""Is My Phone Hacked?" Analyzing Clinical Computer Security Interventions With Survivors of Intimate Partner Violence," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–24, 2019.
[103] E. Tseng, R. Bellini, N. McDonald, M. Danos, R. Greenstadt, D. McCoy, N. Dell, and T. Ristenpart, "The Tools and Tactics Used in Intimate Partner Surveillance: An Analysis of Online Infidelity Forums," in *Proceedings of the 29th USENIX Security Symposium*, 2020, pp. 1893–1909.
[104] K. Thomas, D. Akhawe, M. Bailey, D. Boneh, E. Bursztein, S. Consolvo, N. Dell, Z. Durumeric, P. G. Kelley, D. Kumar *et al.*, "Sok: Hate, Harassment, and the Changing Landscape of Online Abuse," in *Proceedings of the 2021 IEEE Symposium on Security and Privacy*, 2021, pp. 247–267.
[105] D. Freed, J. Palmer, D. Minchala, K. Levy, T. Ristenpart, and N. Dell, "A Stalker's Paradise: How Intimate Partner Abusers Exploit Technology," in *Proceedings of the 2018 CHI conference on Human Factors in Computing Systems*, 2018, pp. 1–13.
[106] C. Fraser, E. Olsen, K. Lee, C. Southworth, and S. Tucker, "The New Age of Stalking: Technological Implications for Stalking," *Juvenile and Family Court Journal*, vol. 61, no. 4, pp. 39–55, 2010.
[107] A. Shimizu, "Domestic Violence in the Digital Age: Towards the Creation of a Comprehensive Cyberstalking Statute," *Berkeley Journal of Gender, Law and Justice*, vol. 28, p. 116, 2013.
[108] C. Southworth, S. Dawson, C. Fraser, and S. Tucker, "A High-Tech Twist on Abuse: Technology, Intimate Partner Stalking, and Advocacy," *Violence Against Women*, 2005.
[109] C. Southworth and S. Tucker, "Technology, Stalking and Domestic Violence Victims," *Mississippi Law Journal*, vol. 76, p. 667, 2006.
[110] M. Dragiewicz, B. Harris, D. Woodlock, M. Salter, H. Easton, A. Lynch, H. Campbell, J. Leach, and L. Milne, "Domestic Violence and Communication Technology: Survivor Experiences of Intrusion, Surveillance, and Identity Crime," *The Australian Communications Consumer Action Network*, 2019.
[111] R. Mayrhofer, J. V. Stoep, C. Brubaker, and N. Kralevich, "The Android Platform Security Model," *ACM Transactions on Privacy and Security*, vol. 24, no. 3, pp. 1–35, 2021.
[112] Vice Motherboard. (2017, 04) Inside the 'stalkerware' surveillance market, where ordinary people tap each other's phones. [Online]. Available: https://www.vice.com/en_us/article/53vm7n/inside-stalkerware-surveillance-market-flexispy-retina-x
[113] E. Pan, J. Ren, M. Lindorfer, C. Wilson, and D. Choffnes, "Panoptispy: Characterizing Audio and Video Exfiltration From Android Applications," *Proceedings of Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 33–50, 2018.
[114] H. Sbai, "The Threat of Screenshot-Taking Malware: Analysis, Detection and Prevention," Ph.D. dissertation, University of Oxford, 2022.
[115] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on WebView in the Android System," in *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011, pp. 343–352.
[116] E. Chin and D. Wagner, "Bifocals: Analyzing Webview Vulnerabilities in Android Applications," in *Proceedings of the 2013 International Workshop on Information Security Applications*, 2013, pp. 138–159.
[117] M. Neugschwandtner, M. Lindorfer, and C. Platzer, "A View to a Kill:WebView Exploitation," in *Proceedings of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2013.
[118] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang, "Identity Confusion in WebView-based Mobile App-in-App Ecosystems," in *Proceedings of the 31st USENIX Security Symposium*, 08 2022, pp. 1597–1613.
[119] A. Pham, I. Dacosta, E. Losiouk, J. Stephan, K. Huguenin, and J.-P. Hubaux, "HideMyApp: Hiding the Presence of Sensitive Apps on Android," in *Proceedings of the 28th USENIX Security Symposium*, 08 2019, pp. 711–728.
[120] Department of Justice. (2022, 02) Pakistani man indicted for selling 'stealthgenie' spyware app. [Online]. Available: https://www.justice.gov/opa/pr/pakistani-man-indicted-selling-stealthgenie-spyware-app
[121] D. McCoy, H. Dharmdasani, C. Kreibich, G. M. Voelker, and S. Savage, "Priceless: The Role of Payments in Abuse-Advertised Goods," in *Proceedings of the 2012 ACM conference on Computer and Communications Security*, 2012, pp. 845–856.
[122] FTC. (2022, 02) Ftc finalizes order banning stalkerware provider from spyware business. [Online]. Available: https://www.ftc.gov/news-events/press-releases/2021/12/ftc-finalizes-order-banning-stalkerware-provider-spyware-business
[123] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: A Text Mining Approach to Analyzing and Classifying Code Structures in Android Malware Families," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1104–1117, mar 2014. [Online]. Available: https://doi.org/10.1016/j.eswa.2013.07.106
[124] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1105–1116.
[125] L. Deshotels, V. Notani, and A. Lakhotia, "Droidlegacy: Automated Familial Classification of Android Malware," in *Proceedings of the 2014 ACM SIGPLAN on Program Protection and Reverse Engineering Workshop*, 2014, pp. 1–12.
[126] Guardsquare. (2022, 06) Leader in mobile app security. [Online]. Available: https://www.guardsquare.com/
[127] MichaelRocks. (2022, 06) Michaelrocks/paranoid: String obfuscator for android applications. [Online]. Available: https://github.com/MichaelRocks/paranoid
[128] Giacomo Ferritti. (2022, 06) giacomoferretti/paranoid-deobfuscator: Deobfuscate "paranoid" protected apps. [Online]. Available: https://github.com/giacomoferretti/paranoid-deobfuscator

Families	App Name	App Website	APK Version	Target SDK	Package Name	Discovery Method
Spyic	Cocospy	cocospy.com	16.3	22	com.sc.cocospy.v2	Website & APK
	Minspy	minspy.com	16.3	22	com.minspy.v3	
	Neatspy	neatspy.com	16.3	22	com.sc.spyic.v3	
	Spyic	spyic.com	16.3	22	com.sc.spyic.v3	
	Spyier	spyier.com	16.3	22	com.sc.spyier.v2	
	Spyine	spyine.com	16.3	22	com.sc.spyine.v2	
	Spyzie	spyzie.io	16.4	22	com.dy.spyzie.v4	
TheTruthSpy	Copy9	copy9.com	7.85	28	com.systemservice	Website & APK & Backend
	GuestSpy	guestspy.com	5.0.1	28	com.systemservice	
	iSpyoo	ispyoo.com	8.80	28	com.systemservice	
	Mxspy	mxspy.com	1.0	22	com.mxspy	
	TheTruthSpy	thetruthspy.com	9.41	28	com.systemservice	
HoverWatch	HoverWatch	hoverwatch.com	7.2.338	28	com.android.core.mntw	Website & APK & Backend
	Snoopza	snoopza.com	6.1.56	28	com.android.core.mngu	

Table 5: App families identified while selecting apps to study. For each app in a family we show its name, website domain, APK version code, target SDK version, package name, and how we discovered the family connection.

A APP OBSERVATIONS

While examining the spyware apps, we observed that they share many characteristics. To supplement our main results, we also describe similarities in their installation configurations and the families of apps we discovered during our app selection process.

A.1 Installation Configuration

Many spyware apps share common installation configurations. These configurations help spyware apps stay stealthy and improve persistence. Broadly, we see the following common configurations recommended by spyware apps: (1) turn off Google Play protection; (2) turn off notification of the app; (3) grant various permissions (e.g., accessibility); and (4) disable battery optimization. Additionally, we note that seven apps (Clevguard, HoverWatch, iKeyMonitor, Meuspy, Spyhuman, Spy24, and TheTruthSpy) abuse accessibility to automatically click buttons (e.g., automatically clicking on the grant permission button when the app is requesting MediaProjection permission), a behavior much like that of malware. Finally, we find that three apps (Meuspy, Mobile-tracker-free, and Spy24) use a loader app to install the actual app, and a loader facilitates the installation process of the two apps (Meuspy and Spy24) that do not have a launcher activity (as mentioned in Section 3.4).

A.2 App Family

In the process of selecting the 14 apps in our study, we identified various families of apps that are connected: apps that are rebranded versions of others. Table 5 lists the three families of apps we observed. The families we identify echo what is documented by other industry reports [18, 19]. We name each family using the name of the app whose website domain has the highest Tranco ranking, as shown in Table 1.

Spyic is the largest family we find, consisting of seven variants. After examining their APKs, we conclude that all apps in this family are rebranded versions of each other with different package names. We note that websites used by these apps all include a CSS file (`alicdn.com/t/font_xxx.css`) hosted at Alicdn (a Chinese CDN), and some of the debug strings are in Chinese. While it is likely that this family of apps is operated by a Chinese-speaking group, it does not seem to operate in China and does not offer Chinese as a language option on its website, potentially due to legal concerns.

TheTruthSpy family includes five apps: Copy9, GuestSpy, Mxspy, iSpyoo and TheTruthSpy. Examining the APKs of Copy9 and GuestSpy suggests that they are simply rebranded versions of TheTruthSpy. Mxspy uses the same backend as Copy9. Snoopza is a rebranded version of HoverWatch, which can be determined by examining its APK (similar code), backend (same infrastructure) and website (generated with the same template).

We also note that Spy24 advocates for Mobile-tracker-free on its website. However, we observe that the two apps are very different in their implementation, and we include both of them in our study.

We end by noting that this list is not comprehensive. When investigating apps to study, our primarily goal was to filter out duplicate apps that are likely rebranded versions of others to avoid redundant work and results. Classifying apps more systematically and comprehensively is a research area in itself [89, 123–125].

B CODE OBFUSCATION

After decompiling all the APKs with `JADX` and examining the decompiled Java code, we observed that two apps (Highstermobile and iKeyMonitor) did not protect their code with obfuscation (i.e., the names of classes, fields, and methods are preserved). Nine apps obfuscated the names of classes, fields, and methods, which could be done easily with tools such as ProGuard [126]. The last three (Meuspy, SPAPP, and Spyhuman) went a step further and also obfuscated strings (potentially to hinder reverse engineering efforts). Among these three, Meuspy used `paranoid` (an open source string obfuscation tool [127]), and we deobfuscated their strings with an open source `paranoid` deobfuscator [128]. We wrote our own deobfuscators for SPAPP and Spyhuman.

App Name	APK Version	SHA-1 Hash of the APK
mSPY	6.3.2	4e675734487baaa93533f5c187c376d37ce28eb3
Mobile-tracker-free	153	e18637c9576a295b02ab3dc8282eb4ca242942dd
Clevguard	4.0.7	e8234174971f4c50964f8b12987f1fa6ce47699a
HoverWatch	7.2.338	33c12f3fbe2b510ceb3111f8198f974040ab05ba
Flexispy	4.16.1	07786dc314f8cab968d0c5a310a71601543bea0e
Spyic	16.3	37ea4d27e3ac25c48b72d99c1503e56853ce7260
Spyhuman	311	f567eff3134b04c0efbc14fa6bc4916bb851ae0c
TheTruthSpy	9.41	d421e9a94c742f80e9ff573b73576eaf1bb8dc25
iKeyMonitor	9.8	2f6f807f2ac1b5a423e006a667db15c3f7229c6d
Cerberus	3.6.9	40345be0287e47224d951cd3644ac0bd7f49e150
Spy24	1.0	324fb89cec42ab67be6b644b62522c89711b53b0
Spapp	16.6	7936fa6cf35cf74b5e156d63849188c28de86d0b
Meuspy	5.20	2eb5bc667a499e44d3c77c9f16c23d278e56e9f7
Highstermobile	3.26	c0d6b1a18a8cc49f7f804451ee992fe0670072ec

Table 6: The APK's version code and SHA-1 hash of the apps we study. Shading added to improve readability.

C ADDITIONAL FIGURES AND TABLES

Table 6 shows the list of spyware of apps we chose, their APK version code, and the SHA-1 hash for the corresponding APK we study. Figure 3 shows an example of an app disabling both the Force Stop and Uninstall buttons on a phone running Android 6.

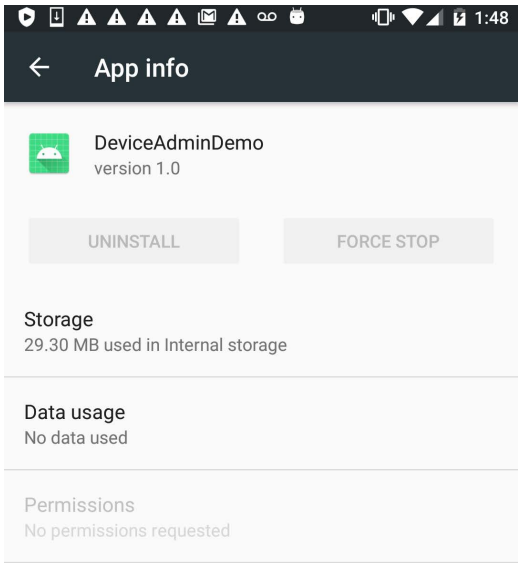

Figure 3: An example of an app disabling both the Force Stop and Uninstall buttons on Android 6.