


# Passwords To-Go: Investigating Multifaceted Challenges for Password Managers in the Android Ecosystem

Nicolas Huaman <sup>ℓ</sup> Marten Oltrogge <sup>ℓ</sup> Sabrina Klivan <sup>ℓ</sup> Yannick Evers<sup>ℓ</sup> Sascha Fahl <sup>ℓ</sup>

<sup>ℓ</sup>*CISPA Helmholtz Center for Information Security*  
<sup>ℓ</sup>*Leibniz University Hannover*

**Abstract**—Android provides two APIs to help mobile apps and browsers interact with password managers, the Android Autofill framework (AAF) and the Credentials API. Mobile password managers rely on these APIs to insert stored credentials into apps and browsers, limiting user interaction during authentication. However, implementing these APIs correctly can be challenging for app developers. For example, misusing the AAF can lead to insecure authentication, login credential phishing, and decreased usability.

In this work, we conduct a mixed-methods study on the use of Android authentication APIs, focusing on their password manager support and impact on authentication security and usability. We first conduct a large-scale analysis of the two authentication APIs in 639,731 Android apps. Secondly, we perform an in-depth qualitative analysis of the AAF with 100 apps, ten browsers, and eleven password managers on Android. The Credentials API has not yet been adopted broadly, illustrating its recent introduction. Regarding Android’s Autofill framework, our qualitative analysis identified various unsupported edge cases like credit card management and password changing. Based on our findings, we make recommendations for improving the AAF and relate them to the Credentials API. We find that while a lot of the partially supported cases will work better in the new API, especially the lesser supported cases in our analysis currently fail for both APIs.

**Index Terms**—Authentication, Android, Password Manager

## 1. Introduction

Password Managers (PWMs) are an important tool to manage passwords and accounts on the web securely. They offer secure storage and generation of passwords, reducing the burden on memorizing and inventing secure passwords for the user. Invented as an afterthought to authentication, however, there are multiple issues that plague PWMs on the web and desktop systems [1], hindering their adoption [2]. On mobile, integrated authentication frameworks step in to solve some of the interaction issues between PWMs and apps, including a consistent interface for all apps that PWMs can hook into and apps can test for in their authentication flows [3]. While past research has investigated usability and security issues of PWMs, none have investigated the causes of these issues in the mobile ecosystem.

We provide two major contributions in this work. First, we present an evaluation of framework prevalence between the two dominant authentication frameworks on Android, the AAF [4] and the Credentials API [5]. Second, we analyze root causes for issues between interactions and the AAF, and uncover limitations that prevent developers from providing good usability and security when using the AAF both as PWM and as app developers trying to obtain credentials through the framework. We find the AAF is the dominant solution for apps and conduct five experiments to gain in-depth insights into its security and usability for developers. Since the AAF provides support for AutoFill services, which in most cases represent PWM, we evaluate it using a similar approach to Huaman et al. [1], identifying multiple edge-cases of interactions and then evaluating them with the stakeholders of the AAF. We identify 63 cases in which the AAF can create usability issues for end-users and test their prevalence across 100 top apps and ten browsers. In this paper, we investigate the following research questions:

**RQ1: How many Apps use each authentication API?** We analyze 639,731 Android apps to investigate their use and the prevalence of both authentication APIs, the AAF and the Credentials API.

**RQ2: What limitations do PWMs encounter when using the AAF?** We investigate the options and limitations for PWMs and apps when interacting with the AAF regarding their security and usability by performing two qualitative experiments, identifying and testing edge-cases in interactions between apps, the AAF and PWMs.

**RQ3: How can stakeholders of the Android ecosystem better support the interaction between PWMs and apps?** We derive recommendations from our observations regarding unsupported use cases for the AAF on Android. These recommendations include focus points for Google as the provider of the Android SDK, PWMs and app developers, and notable mentions for considerations the Credentials API as a successor needs to deploy.

To answer these research questions, we performed a mixed-methods study. To the best of our knowledge, this work is the first in-depth analysis of mobile PWM support on Android and the root causes of usability and security issues. We provide the following contributions:

**Analysis of Android Apps and their Authentication Support.** We perform a large-scale static analysis of 639,731 apps, to investigate their use of properties and APIs relating to the two authentication APIs (cf., Section 3). We find that the new Credentials API lacks the adoption required to analyze its use in meaningful ways.

**Manual In-Depth Analysis of the AAF on Android.** We analyze the interaction between Android apps and PWMs using the AAF [4]. In multiple qualitative experiments, we collect information about the interactions between PWMs and apps, covering ten different browsers and 100 apps (cf., Section 3). We identify 63 categories of problems across all parts of the chain, causing issues with field detection, app compatibility, and block/allow listing-breaking compatibility that affect the usability of the framework, and we uncover fundamental issues like missing authorization for domain usage in the AAF, which lead to block/allow listing by PWM becoming necessary to prevent unintentional security leaks despite the usability issues it causes.

**Replication Package and Recommendations.** Based on our results, we offer suggestions to improve the existing state of affairs to address the limitations we encountered. Section 6 provides an in-depth discussion of our findings and recommendations for developers and the AAF. We also talk about the new Credentials API and how well it solves the issues we identified for the AAF, finding unsupported use cases and a general need for more integration with the AAF. For research transparency, we provide a detailed replication package [6]. This package includes all code, test cases, and additional information for replication.

## 2. Related Work

We discuss related work in three key areas: Security and usability of mobile PWMs, and alternatives to password authentication.

**Password Manager Security.** In this section, we present research analyzing the security of PWMs, especially regarding the AutoFill and AutoSave mechanisms that are most relevant for mobile PWMs. First, Silver *et al.* investigated the security of AutoFill features for multiple PWMs, finding exploits in their AutoFill policies. As a solution, they proposed to force user interaction to select and fill passwords, and server-side defenses like CSP, HTTPs, and externalizing the login page to limit the effect of same-site attacks [7]. Aonzo *et al.* presented a case study of vulnerabilities in common PWMs’ package matching strategies [8]. They proposed a secure-by-design API for the AAF and stronger reliance on asset links to mitigate vulnerabilities related to PWMs’ insecure use and limitations of Android accessibility and AutoFill frameworks. Lee *et al.* presented an analysis of the Android frameworks and apps, including PWMs. They found that some sensitive data, such as passwords, remained in memory after use, making them susceptible to memory attacks [9]. Lin *et al.* uncovered new attack vectors using hidden input fields for exfiltration of PII (e.g., phone numbers, credit card data) using AutoFill features in

browsers and presented a large-scale web study measuring the use of these attack vectors on 100,000 sites. They showed that 5.8% of sites contained hidden input fields in ways that allowed them to collect PII, compromising users privacy [10] stealthily. Oesch *and Ruoti* replicated previous studies and evaluated the security of 13 PWMs, and conducted a novel evaluation of password generation features. Finding several potential issues within generation, AutoFill and storage, they recommended that PWMs adopt automatic discarding of insecurely generated passwords, better policies for a PWM master Password and manual confirmation for AutoFill requests to protect users from XSS attacks [11]. Later work by Oesch *et al.* dealt with shortcomings and limitations in Android and iOS AutoFill features and how they make PWMs vulnerable to credential theft by malicious apps [12]. Li *et al.* investigated five web-based PWMs. They identified and presented vulnerabilities allowing data exfiltration via XSS or CSRF attacks, websites embedded in iframes, and insufficient authorization mechanisms in the PWM AutoFill process [13].

The works covered in this section identified different sources of vulnerabilities in PWMs and the web and potential solutions through changes to authorization protocols. In contrast, we focus on inherent usability issues within mobile PWMs, which pose fundamental obstacles regarding interaction between PWMs, apps, AutoFill APIs, and browsers.

**Password Manager Usability.** In the past, several user studies regarding PWMs usability have been conducted, ranging back to 2006 when Chiasson *et al.* evaluated the usability of two PWMs and showed significant usability problems causing mental model mismatches for users on whether their passwords were protected and reducing the willingness to adopt PWMs [14]. Lyastani *et al.* investigated the impact PWMs have on users’ stored passwords. They found that password security increases for PWMs users, but the extent depended on the implementation and usability of the password generation and entry processes [15]. A user study by Seiler-Hwang *et al.* observed the usability of mobile PWMs and also found participants to report AutoFill as problematic [16]. Zhang *et al.* conducted a semi-structured interview study with 30 participants and provided an overview of user sentiments regarding the adoption and usage of PWMs and password-generation features [2]. This study was replicated by Ray *et al.* with a focus on older participants, where they found that an older population encountered more issues with PWMs and generally relied on family recommendations when adopting PWMs [17]. On the web, Huaman *et al.* analyzed PWM usability by testing 15 PWMs on 39 websites with specific behavior to determine error sources when using PWMs. They found faulty interactions in eight areas, including domain handling and input field mislabeling [1]. In another study, Oesch *et al.* identified theories describing users’ rationale behind their PWM usage, by which they also identify any inconsistencies regarding AutoFill and AutoSave functionality on desktop and mobile as reported by participants [18]. Mayer *et al.* surveyed PWM usage at a private university, investigating adoption issues

for faculty, staff, and students [19]. Interestingly, regarding PWM interface design, Zibaei *et al.* conducted a user study to assess the efficacy of nudges and hints browsers provide as part of their internal PWMs for choosing secure passwords [20]. They found that Safari’s built-in PWM provides the most effectively adopted password nudges. Zibaei *et al.* analyzed multiple types of nudges inspired by safari in a follow-up study. They found that a simple default nudge of a strong password filled into the password field has the highest adoption rate for users [21]. Finally, Amft *et al.* evaluated the credential management strategies of users newly adopting PWM and compared them to longer-time users of PWMs. They found misconceptions among their participants, not trusting the PWM to keep personal or important passwords, as well as a general tendency towards convenience, leading users to not rely as much on the secure password generation feature of PWMs [22].

Overall, previous related work on PWM usability was focused on desktop or web-based tools. In contrast to the usability measurements of previous end-user-focused efforts [2], [16], [17], our work examines mobile password management and the technical causes of individual issues. Three experiments of our second study use interactions inspired by Huaman *et al.* [1]. We derive specific interactions for mobile apps from our other experiments and extend their interactions with new variants to test the AAF. To our knowledge, we are the first to identify root causes for usability issues in the AAF.

### 3. Status Quo Analysis

In the following section, we present our large-scale measurement, investigating the presence and integration of these features within in 639,731 freely available Android apps on Google Play. We further investigate how frequent apps use both the AAF [4] and the Credentials API [5]. Sadly, we find that the Credentials API is too recent to be found and meaningfully used in any apps. Therefore, we focus on the AAF and present a few smaller case studies investigating the usability and related issues of this framework when interacting with, e.g., popular Android apps and browsers.

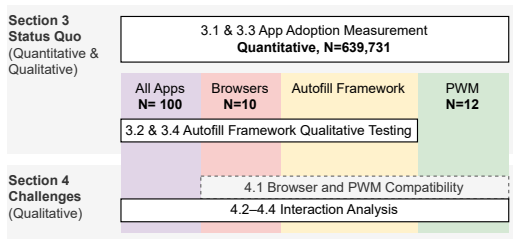


Figure 1. Overview of the experiments for RQ1&RQ2.

#### 3.1. App Adoption Measurement

We measure AAF adoption of apps from Google Play (EEA country location-based) based on the list of apps

from AndroZoo [23], which is a scientific and actively updated dataset of Android apps, commonly used in Android research [24], [25]. Our sample of apps generalizes to the Android ecosystem (cf., Section 5) and mirrors the adoption of the AAF on Google Play.

To analyze our apps, we investigated their APKs, which consist of the compiled source code and include resources and metadata like the APK manifest, which provides information about all resources, views, and permissions used by an app. Compiled source code, which is contained in \*.dex files. Resources are contained in the res/, including images, strings, translations, and other static resources used throughout the apps’ lifecycle. Using Androguard [26], we disassembled the compiled source code and decoded all resource files to collect our desired information.

The following provides an overview of the in-depth features we examine and how we collect them.

**AutoFill Framework Attributes.** The AutoFill Framework is the older one of the two APIs. It is integrated into Android and works by default with all types of inputs and views provided by the Android SDK. Developers therefore only need to address it to fix bugs or make it work with native code and other code circumventing the Android SDK that they provide with their app. There are two ways to address the AutoFill Framework, attributes in XML and API calls. To collect the attributes of the AAF that help PWMs understand which fields are related to authentication, we collect res/layout/\*.xml files in all apps. These files define UI-components like layouts in Android. We then parsed the XML tree of all XML files. To complete, we used XPath to search for elements with android:autofillHints or android:importantForAutofill attribute. We collected both the attribute type and the set value, for example the concrete hint password, as string. Furthermore, developers can also use resource identifier references [27], which work similar to links for value-strings. We resolved all of these links using the remaining resource files to get the value set by the developer.

To address obfuscation [28], we also parsed XML files on potentially obfuscated paths. Thus, we analyzed all \*.xml files to address views declared outside res/layout/.

**AutoFill API Calls.** Second, we collect the API calls that set these properties for views programmatically via View class’ setAutofillHints and setImportantForAutofill methods [29], [30]. To capture these API calls, we performed an in-depth call graph analysis for 639,731 apps. We searched for the above API calls, and collected calls to measure the use of these APIs. Due to the difficulty of obfuscation in the code and the added effort of parameter backtracking for the values of these API calls, we could not collect the values of these calls.

**Credentials API Calls.** The Credential Manager API and Credential Provider API, which we summarize as Credentials API for short, allow developers to directly interact with credential providers, like PWMs to skip the step of filling input fields or similar [5]. Unlike the AAF, it supports other kinds of providers like passkeys and social logins as well.

However, apps need to actively implement this API for it to work. This requires more effort to use because developers need to integrate this API into their apps manually, but reduces the risk of interaction failure. For this API, we perform the same in-depth call graph analysis for 639,731 Apps, looking for the `androidx.credentials` namespace in- and outside the main package of each app.

### 3.2. AutoFill Framework Functional Testing

Since static analysis cannot capture dynamic app behavior, we conducted several qualitative case studies to gain insights into how apps use AutoFill features. We consider three stakeholders involved in AutoFill interactions: Android apps, the AAF, and PWMs (cf., Figure 1). We first performed three case studies to examine the interactions between the Android apps and the AAF currently. Our initial tests indicated that the requirement to adhere to web standards further complicates AutoFill integration, in line with the findings for websites by Huaman *et al.* Therefore, two of our case studies focus on specific types of Android apps, namely browsers and WebViews, which are in-app browser segments used to display websites [1]. In the following paragraphs, we give some context on the specific apps, browsers, and PWMs chosen for our case studies throughout the paper.

**Generic App Selection.** Regarding our sample of generic apps, we chose to analyze recently rated apps on Google Play with many installs, since these represent widespread apps with an active user base. We used Androidrank [31] to inform our app selection. They provide statistics about the Google Play Store, including installs in buckets, reviews, and rating history. On Androidrank, we selected the 148 apps with at least 500 million installations. From this list, we selected the 100 apps for our experiments that received the most ratings in the last 60 days, which lead to all apps having received at least 28,069 ratings. A complete list of all tested apps is available in our replication package 1.

**Browser Selection.** In our first case study (cf., Section 3.4), we found a behavioral difference between logins in a WebView and normal apps. We also investigate different browser implementations to address this difference and gain insights into the causes. Similar to apps, we chose the most installed browsers that appeared in the Play Store when searching for “browser”, leading to 28 apps with at least ten million downloads. Since browsers share engines, i.e., they are commonly based on Google’s Blink engine or WebKit, we assume many act similarly. We provided a small testing website with only login and password fields to test this assumption. We categorize the 28 browsers into the following four levels: First, No support at all, not even for legacy hacks. We excluded these browsers since they do not work whatsoever for any experiments. Second, Support in Compatibility Mode, which means support using a legacy accessibility feature not intended for PWMs. These browsers notably include the Google Chrome Browser, which has no support for third-party PWMs and manually relies on

Google Cloud credentials. The third level, “Native Support” refers to browsers that do not disable the AAF and therefore support third-party PWMs. Examples are the Opera Mini and the Mi Browser Pro. Finally, our testing revealed as a final tier that the Gecko browser engine includes a sophisticated translation layer to provide valid AutoFill hints to the AAF based on detected fields. An overview of all browsers, categories, and our browser selection can be seen in Table 8.

**Websites for Testcases.** To test browser and website support for the AAF, we started with the set of testing websites for PWMs from previous work [1]. Since these are targeted at desktop or laptop instead of mobile interaction, we adapted and filtered some of their interactions. Any tests requiring button interactions, copy and paste, or manual clicks were excluded, since the AAF does not provide any direct interaction beyond filling the input fields. Furthermore, we excluded all but one of the domain test cases because we found that in all AutoFill requests, PWMs can only see the domain, not the full URL. In addition, we added tests with variants of the autocomplete attribute mapped to the Android AutoFill default values, since many browsers in our previous experiment did not translate these hints. We provide an overview of our testing websites in Appendix B.1.

**Testing Apps with the AutoFill Framework.** Due to the large number of apps, we tested 100 apps in the default Android developer tools emulator [32], the recommended way to test Android apps device-independently. We captured the type of credential information our apps exposed through the AAF on their login screens, and all errors. To receive the provided information, we used a stub AutoFill service, as well as the UIAutomatorViewer [33], which is part of the Android SDK manager, to inspect the implementation of input fields in views. For each app, we checked whether it contained a valid login or similar view that allowed the input of credentials inputs or other data such as credit card information, phone numbers, or addresses. We collected AutoFill hints and field IDs when encountering such a view in our stub AutoFill service. IDs are unique identifies for text fields in apps. Using the ID of a password field in a popular app, PWMs can provide reliable autofilling in cases where the AutoFill hint attribute is missing. The behavior and edge cases detected through this experiment serve as a baseline for the interactions we designed in the following experiments.

**Testing Websites with the AutoFill Framework.** Websites parsed by browsers and WebViews add another layer of complexity to the interaction with the AutoFill framework. We investigated the compatibility of popular Android browsers with the AAF with our stub AutoFill service. This includes collecting how they report AutoFill hints, and how they identify domains they want to transfer. We therefore created test websites, and examined what type of information browsers provide to PWMs when visiting these websites. We provide an overview over our test websites in Table 10. We tested all previously selected browsers in this experiment (cf., Table 8) on a Realme 8 (API 30).

Using the AAF and our stub AutoFill service, we recorded all information provided by the browsers.

### 3.3. Results of the App Adoption Measurement

In our large-scale adoption measurement, we collected indicators for the adoption of the AAF and Credentials API. We provide our insights in the following.

**AutoFill Framework Usage.** Since the AAF is embedded into Android, it works with all apps by default, requiring no manual implementation. We therefore look for explicit interaction with the AAF to grasp app usage.

Thereby, our focus is on explicit use of the AAF framework in layout XML files and API calls by app developers, meaning that we only deem API calls relevant that are located in apps’ main code package rather than e.g., library code. In total, we infer the AAF to be used in 149,325 apps (23.34%). For 148,898 apps (23.28%) we found they utilize the XML notation, and in 825 apps (0.13%) we detected use of the API in source code. This provides an upper bound for developer interactions with the AAF.

Overall, interactions are rather low. This suggests that despite its ubiquity within Android app development, only a few developers actively engage with the AAF.

API calls are particularly low, with only 0.13%. Within these, 734 apps (0.11%) make use of `setImportantForAutofill` and 452 apps (0.07%) actively set `setAutofillHints`. This indicated that options and hints for the AAF are predominantly declared in view XMLs, not in app code.

**Reasons for Usage.** For the 148,898 apps that use AAF by XML notation, we also collected explicit AutoFill hints in 47,676 apps (7.45%). Table 1 depicts the distribution of top 5 AutoFill hints in `android:autofillHints` we found among apps declared in XML, and the presence and usage of `android:importantForAutofill` attributes. Unsurprisingly, password field hints were the most common, with blank hints being the second most common. We suspect these blank hints were copied and pasted from examples or documentations, but never filled with. In contrast to the rather low usage of `android:autofillHints`, we found `android:importantForAutofill` to be the most common AutoFill-related feature present in 124,514 apps (19.46%).

While the presence of AutoFill hints, in general, indicates an explicit usage of the AAF, we find that most of the `android:importantForAutofill` attributes are in practice used to mark views irrelevant for AutoFill, thereby effectively “hiding” them from PWMs. An explanation for this could be an old recommendation for the deprecated framework Smart Lock [36] to disable the AAF, but it could also point at general issues when interacting with the AAF.

Since the AAF is the Android default, we assume most interactions to be targeted at either supporting it or fixing bugs that occur with it. Therefore, the low interaction rate we collected indicates that it works fine or is ignored by the majority of apps. The high number of deactivations among the apps that interact with the AAF does indicate

TABLE 1. PREVALENCE OF ATTRIBUTE VALUES FOR IMPORTANTFORAUTOFILL [34] AND AUTOFILL HINTS [35].

importantForAutofill	# Apps
View not important (0x02)	121,519
View and descendants important (0x01)	4,520
View and descendants not important (0x08)	2,945
Auto (0x00)	474
View important, descendants not (0x04)	237
Combination: 0x02   0x08 (0x0A)	13
AutoFill Hints	# Apps
password	8,297
<blank>	5,913
username	5,882
emailAddress	4,243
postalAddress	3,982

TABLE 2. AVAILABLE LOGIN OPTIONS ACROSS OUR APP DATASET.

Login	Apps	Ratio	AutoFill
Google (Play Games)	46	54.76%	No
Facebook	23	27.38%	No Hints
Twitter	5	5.95%	No Hints
VK	4	21%	No Hints
Login Fields	50	59.52%	Custom

potential issues that need to be investigated, and AutoFill hints could help PWM. We elaborate how the situation could be improved for the AAF in the case studies after our large-scale analysis.

**Credentials API.** The newer Credentials API replaces a few older APIs and adds support for third-party PWMs, making it a suitable candidate for our investigation. However, we only found this API in 52 apps, representing an adoption of 0.01% across our dataset. With 39 apps (0.01%), even fewer apps use the API in their main package, indicating that active usage is pretty limited. Manual investigation further revealed that the framework is overwhelmingly used for passkey support without enabling PWMs. We suspect that adoption will pick up as apps update and PWMs adopt this new API, but due to lack of adoption, we excluded the API from our qualitative analysis, since a representative sample is not yet available.

### 3.4. Results for Functional App Testing

For our qualitative case study regarding apps, we excluded eight apps unsuitable for testing, including keyboard apps, background services, and others that users cannot directly launch. Eight more apps either used the native framework or prevented analysis via UIAutomatorViewer, making assessing these apps impossible. We therefore considered 84 apps in our analysis. Six apps failed to run on the emulator, so we tested them on a Realme 8 (API 30) similar to browsers.

**Third-Party Logins.** We examined the login methods of these apps and provide an overview in Table 2. Here, the most common method aside from service-specific logins is

TABLE 3. DISTRIBUTION OF APP LOGINS AND TYPES.

AutoFill Hints LoginType	Yes	Custom	No	No Autofill
WebView		2	6	
Service-specific View				3
Standard View	3		29	6
Not Definitive				1
<b>Total</b>	3	2	35	10

Google Play Games with 46 apps (54.76%). This method is integrated into Android and used by many games to ease account management and allow for Android-integrated achievements and social features. When a Google account is already set up, this method works automatically. It requires no AutoFill at all, but when credentials have to be provided to sign in to a new Google account, AutoFill using installed PWMs is unsupported. When logging in, Facebook, Twitter, VK and other social logins open a WebView, which supports AutoFill.

**Webviews and Service-specific Logins.** Regarding service-specific login forms implemented by app developers, 50 apps (59.52%) in our dataset provided a view with login or credential fields that could be suitable for a PWM. Within this set, only five (10%) provide hints on their input fields. On the other hand, ten apps (20%) do not trigger an AutoFill request at all. Upon closer investigation, we found this to be related to the respective app’s type, splitting them up into three distinct categories: Web apps that use a WebView to display their components, native or modified views that implement different view logic, and, therefore, need to re-implement the AAF as well, and apps using the standard views. For one app, we could not identify the definitive type of login used. While the app relies on mostly native views, and we suspect the login form to be native, parts of the login fields could not be tracked to their source in our source code analysis. Fittingly, this app did not support the autofill framework at all in our testing. A full overview of our identified support counts for the 50 apps (59.52%) can be seen in Table 3

38 apps (45,24%) use the standard view implementation for their credential forms. This means that support for the AAF is included, and only AutoFill hints are missing. In our dataset, three apps (7.90%) actually provided AutoFill hints. For ten apps, we could not interact with the credential fields through an AutoFill service. Analyzing the source code of these apps, we found that two actively marked these credential fields as “notRelevantForAutofill” despite these credentials being a perfect fit for PWMs. One app used the Firebase UI [37], [38], which does not support the AAF in its credential fields. Upon closer inspection, Firebase actively disables the AAF when Smart Lock, an older API replaced by the Credentials API is active, in line with Smart Lock documentation [39]. Since Smart Lock is active by default, these apps will not support the AAF for third-party PWMs. According to a discussion [40], this is intentionally chosen not to annoy users with multiple pass-

word prompts. Further eight apps (16%) are implemented as WebViews. Android’s default WebView implementation supports the AAF. Chrome WebView, an alternative variant deployed with the Google Play Services, does not provide autocomplete attributes. Finally, we found three apps that implement native or modified views. These apps must implement interfaces and API calls for the AAF manually, which none provided.

**Assistance via IDs.** Regarding the 40 apps that did trigger AutoFill requests, 21 provided IDs from which PWMs could infer what type of credential to provide, as they included English words like “user”, “mail”, “phone”, or “password”. The remaining apps did not define IDs, use generic IDs like “text\_field”, or assign a string of random characters as ID, making them difficult to identify for PWMs.

### 3.5. Results for Functional Website Testing

Finally, we present our evaluation of browsers on Android, and how they interact with the AAF. For brevity, we summarize the results in categories and focus on interesting failures:

**Base Tests.** Regarding input field information, no compatibility mode browser provided any information or attributes, not even a field ID. This means that PWMs needs to make more assumptions and can potentially be misled. Firefox does not invoke the AutoFill framework on two-page logins. Interestingly, Firefox also provides input fields of type “submit” as fields to the PWM. Filling these sets the value and, with it, the text of the submit button, which is likely a bug of the AutoFill implementation. Regarding native browsers, the Mi Browser Pro hands over an attribute called `ua-autofill-hints`, with Opera Mini providing a similar attribute. While these attributes were empty or `NO_SERVER_DATA` for our test websites, they contain hints for popular websites provided by the browser. The functions for the attributes can be found in the Chromium source code [41].

**Domain Tests.** Regarding insecure HTTP websites (D04), the native Chromium-based browsers did not trigger an AutoFill or AutoSave request, but the compatibility mode browsers and Firefox did. For native browsers, this is likely a security measure to prevent clear text transmissions. When iframe domains differed from the main page (D06), no browser transmitted both domains to the framework, but either only the main page URL in compatibility mode browsers or only the iframe domain for native browsers and Firefox. Both approaches can lead to mismatches since all domains across iframes must select credentials.

**Input Field Tests.** All browsers, except Firefox, triggered AutoFill and AutoSave requests in all cases. However, for native browsers and Firefox, no AutoSave was triggered when the username field was a textarea (I06). Furthermore, Firefox did not trigger AutoFill for fields without a type (I05/I07). For password changes (I04), Firefox set the AutoFill hint “password” instead of “newPassword”, preventing PWMs from categorizing this correctly.

**Additional Field Tests.** Regarding the interaction with an additional “Last Name”-field (AA04), Firefox did not recognize it, not even with autocomplete attributes (AA04a). When encountering additional radio buttons and checkboxes (AN02), all browsers transmitted those to the AAF. For browsers in compatibility mode, the AutoFill type is `NONE`, despite the existence of a `TOGGLE` in the framework. Firefox provides the AutoFill type `TEXT`, making these checkboxes indistinguishable from normal text boxes. AutoSave of additional fields worked for no browser in our dataset. The method “`isChecked()`” that is part of the AAF always returns false for checkboxes in all browsers. Finally, if the page contains no login fields but address, phone number, and credit card information (AN06a), all native support browsers except Firefox trigger AutoFill requests with the additional information. Here, the `ua-autofill-hints` attribute contains hints like “`NAME_FIRST`” as part of the address. Since these are also part of the undocumented hints [41] from our base tests, they must be hard-coded by PWMs.

**Basic Auth Tests.** Basic Authentication (W01) is a particular case since its implementation differs between browsers. The browser generates the dialog field, not the HTML rendering engine. Despite this, all browsers send requests to the AutoFill service. However, only Firefox and Samsung Internet Browser transmit the domain for the Basic Auth request, and only Firefox provides AutoFill hints for the username and password field of the request. We encountered an edge case with Opera, where accessing a standard form from our previous interactions before W01 led to the browser suddenly transmitting the previous domain and all previous fields as part of this Basic Auth request, even when navigating to a different subpage. This happened irregularly and was not always reproducible, so we suspect a bug in the browser.

**Non-Standard Form Tests.** Our non-standard interaction N01 contained input fields with no form element and instead AJAX validation. Only compatibility mode browsers triggered an AutoSave request when sending new information.

**JavaScript Tests.** If parts of the form require key presses to turn visible (J01), only native browsers successfully detect them after activation. Both native browsers and Firefox provide invisible fields in the AutoFill request. While native browsers retain the information that these fields are invisible, Firefox erroneously marks them as visible. There are more differences for two-step logins, which provide the password field only after sending the username (J03a). Firefox does not provide an AutoFill request for single fields. Other browsers re-trigger the AutoFill requests when typing into the password field after entering the username manually. While native browsers only fill the password during this second request, compatibility browsers also re-insert the username. When testing AutoSave for this case, for compatibility browsers, we found that the username is not added into AutoSave requests, leading to a mismatch between what users see and what is saved. For native browsers, when users edit the username, instead of overwriting existing data with the edited username, the browsers save the previous

credentials again. The final JavaScript test consisted of a dummy password field swapped with the real one when the user started typing. All browsers, except Firefox, mark the password field as an invisible field, so it is left empty.

## 4. Password Managers and the AAF

After investigating apps’ and browsers’ use of the AAF, we test the edge cases we found and investigate what challenges PWMs encounter because of them. Similar to our previous split, we decided to split between cases that apply to apps in general and cases that focus on browsers and WebViews specifically.

**Password Manager Selection.** We selected the eleven most popular third-party PWMs. To determine these, we searched for the terms “password manager” and “password autofill” in the mobile version of Google Play<sup>1</sup>. The “password autofill” covered apps that support the AAF, but do not describe themselves as “password manager”. Overall, we found 321 apps. Based on our results, we consider these search terms exhaustive. With 50+ million, our dataset’s most downloaded PWM was Microsoft Authenticator. We only considered popular PWMs with at least 1 million downloads, leading to a total of 23 relevant apps. Upon manual inspection, five of the 23 apps turned out to not be PWM apps, but unrelated and unsuitable apps. Furthermore, based on our focus on the AAF, we removed PWM apps that used other frameworks. Our final list of PWMs encompasses eleven apps suitable for our analysis. We provide an overview of our search results and the selected PWMs in Table 7 in the Appendix.

### 4.1. Browser and PWM Compatibility

Due to a lack of verification abilities of the AAF, browsers and PWMs need to employ block- and allow-lists for apps they want to interact with [42]. We measure compatibility between our Android browser selection and several PWMs for this experiment. We tested the ten browsers with the eleven PWMs we selected for this analysis (cf., Section 4) using the same setup as previously on a Realme 8 (API 30). An account for the PWM was created where necessary. Each PWM was equipped with login data for our test pages. Each PWM was set to be the active AutoFill-Service when we tested it. When testing, we went through each browser and examined whether AutoFill and AutoSave worked as expected. We used a basic test involving a simple login page with HTML autocomplete hints (see S01a in our interaction Table 10 in the Appendix), since we regard this as the most standard interaction situation regarding, e.g., default settings and completeness of information. We cover less ideal conditions, our more sophisticated experiments, in Section 4.1 and 4.2. Table 4 shows an overview of compatibility results. In the following, we will discuss the results and their implications:

1. The Google Play desktop website only reveals a selection of results

TABLE 4. COMPATIBILITY BETWEEN BROWSERS AND PWMs.

	Framework	Microsoft	LastPass	Keeper	Dashlane	IPassword	Kaspersky	Keepass2Android	Norton	SafeInCloud	Blackberry	Bitwarden
Google Chrome	C									*	*	
Samsung Internet Browser	C									*	*	
UC Browser	-											
Opera Mini	N											
Firefox Browser	N							*			*	
Opera Browser	-											
Mi Browser Pro	N											
Phoenix Browser	N											
Dolphin Browser	N											
Brave Private Browser	C											

**Compatibility.** Regarding the browsers without support for the AAF in the previous test, we found that no PWM could interact with them either. Surprisingly, the Samsung Internet Browser, which only supports the compatibility mode of the AAF, was compatible with just two PWMs. This is likely a disadvantage of the whitelisting approach. More in-depth, for Microsoft Authenticator interacting with Firefox, we noticed that Firefox did not always succeed at filling in selected credentials (Step 5 in Figure 2). After some investigation, we attribute this to Microsoft Authenticator pausing Firefox behavior. When returning to the Website in Firefox after selecting credentials, the browser reloads the page, and AutoFill may fail because the login fields are not yet reinitialized. The issue does not occur with other browsers. Furthermore, we found Dashlane and Norton to be incompatible with specific browsers. Decompiling revealed that at least one of our browsers was not on the allow list for these PWM. Noticeably, Norton was the only PWM with the Samsung Internet Browser in its allow list. Kaspersky is incompatible with most of our browsers. Upon decompiling the app, we found that it has no list of apps with compatibility mode, leading to this PWM not supporting our first category of browsers. Compatibility here seems to be entirely solved through the Accessibility Service, despite the Google Play Policy [43].

**Inconsistencies & Bugs.** We observed a few inconsistencies independent of interactions in our testing sequence. To verify this, whenever we found an unexpected result in a test with one PWM, we performed the test twice for all PWMs and noted inconsistencies, workarounds, and their causes. Inconsistent results are marked (\*) in our table. First, when testing Firefox with Bitwarden, the AutoFill suggestion was not consistently displayed, which can be fixed by temporarily switching to another app. Second, Dashlane and Norton did show AutoFill suggestions in Opera Mini but did not fill in credentials after selecting them. A restart would usually fix these. Third, when using LastPass, Dashlane, Keepass2Android, or Bitwarden, clicking on the URL bar in Firefox regularly freezes the browser. Finally, multiple clicks on input fields were sometimes necessary across all

browsers and fields to trigger an AutoFill prompt.

## 4.2. Interaction Analysis

To evaluate how well Android PWMs apps interact with Android apps, we constructed 26 interactions in total.

**App interaction with PWMs over the AutoFill Framework.** For this experiment, we tested interactions between apps and PWMs through demo views that supported the AAF to varying degrees. These interactions can roughly be grouped into the following three categories: First, a login view with two fields: username and password. Second, a two-step login spread across two views. And third, additional information, including 2FA, credit card data, or phone numbers. For each of these, we created variable test views containing credentials with different combinations of AutoFill hints, popular ID names we identified before, a clear field type, and a description for the field. In addition, we created “saving” tests, examining whether PWMs successfully stored new credentials. Sensible combinations of these factors resulted in a total of 26 different considered interactions. A complete overview of these interactions can be seen in the Appendix Table 9. We used the eleven PWMs we selected previously and evaluated them in an emulator with a Pixel 4, Android API Level 30, x86 setup. Since we noticed discrepancies between the AAF and the Google One Tap Framework, we included the Google AutoFill service with Smart Lock [39] as an additional PWM. Smart Lock was a Google PWM-only API that did not use the AutoFill Framework, and has now been deprecated, but is included here for comparison to AAF, since the Credentials API equivalent was not yet ready at the time of testing (cf., Appendix A). The PWM of Kaspersky blocks running in an emulated environment. Therefore, it was tested on a Realme 8 (API 30). To equalize testing, we mirrored the Realme 8 screen using screpy [44].

**Browser interaction with PWMs over the AutoFill Framework.** We built websites for each test case, excluding tests that did not trigger AutoFill requests (cf., Section 3.5, and Table 9 in the Appendix, e.g., N01). Testing of Browser and AAF interaction also revealed that some browsers directly transfer autocomplete attributes without converting them to the default AAF variables. We, therefore, added new “-aa” test cases simulating the effect of a translation from autocomplete attributes [45] to Android AutoFill default values. Table 10 in the Appendix contains a complete overview of all interactions for this experiment, including test setup and results. Our test was conducted within the same conditions as the previous experiments on a Realme 8 (API 30). We set each PWM as AutoFill service and then tested each interaction with all of them, aiming to store default username and password values, and credit card, address, and phone number where necessary (AA04 & AN06). As PWM, we used our pre-selected set of PWMs together with the three most popular browsers from each of the three AutoFill-compatible categories: Chrome, Firefox, and Opera (cf., Section 4). Due to issues with browser allow

lists provided by PWMs, we used the most popular allowed browsers with the same engine as replacements for two PWMs, i.e., the Mi browser for Kaspersky, and Samsung for Norton (cf., Section 4.1). We also excluded the Google AutoFill Service for these tests because the browsers use Google’s built-in APIs to access credentials for this service when no other PWMs is enabled.

### 4.3. App Interactions with PWMs over the AutoFill Framework

After testing all scenarios with all PWMs, we compiled an overview denoting what interactions succeed, which we will detail in the following. We summarize all the results for each category we looked at for brevity. Table 5 shows our results for all interactions.

TABLE 5. TEST RESULTS FOR OUR APP TESTING.

Test Description	# Successful	Microsoft	Bitwarden	LastPass	Keeper	Google	Dashlane	IPassword	Kaspersky	Keepass2And.	Norton	Safeincloud	Blackberry
<b>Basic Login</b>													
L0 Save login	10	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L1 Fill login	12	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L2 No AFHints	12	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L3 No unique IDs	9	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L4 No description	8	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L5 Cleartext, AFHints	11	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L6 Cleartext, no AFHints	10	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L7 WebView local	11	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
L8 WebView external	8	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
<b>Two-Step Login</b>													
P0 Save password	4	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
P1 Fill password	12	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
P2 No AFHints	12	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
P3 No unique IDs	11	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
N0 Save username	1	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
N1 Fill username	11	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
N2 No AFHints	8	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
N3 No unique IDs	0	Failure	Failure	Failure	Failure	Failure	Failure	Failure	Failure	Failure	Failure	Failure	Failure
Z0 2step, 2 Activities	1	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
Z1 2step, 1 Activity	6	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
<b>Additional Information</b>													
K0 Save credit card	1	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
K1 Fill credit card	2	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
K2 No AFHints	2	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
T0 Save phone number	1	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
T1 Fill phone number	3	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
T2 No AFHints	3	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success
T3 No unique IDs	1	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success	Success

**Basic Login.** Most of the scenarios are well-supported by most PWMs, even when no description or ID is available, as long as two input fields are available, with one being of type password. When a password field was missing, and no ID or descriptive label was provided, some PWMs could not fill the form. Furthermore, when the login is contained in an external WebView, four PWMs would not offer any AutoFill, likely related to security.

**Two-Step Login.** When logins were spread across two pages, PWMs struggled a lot more. For the single username

field, almost no PWM could save the username, while storage was at least offered for single fields of type password. AutoFill worked similarly well but failed in scenarios with insufficient hints. In case of multiple views or activities, six PWMs were able to store credentials when the same activity provided both views, only one PWM would allow storing credentials for a login split across two activities.

**Additional Information.** When testing for additional information fields, we see a lot more variance between PWMs. Saving credit card information is only possible with the Google PWM, and only with a real credit card. The card is then added as a payment method to the Google Account, not just stored in the PWM. Three PWMs stored parts of the credit card information, one explicitly excluded the security code, which could be considered a security feature. Two PWMs can fully fill stored credit card information, while four partially filled them, one only if provided with a corresponding AutoFill hint. In two cases, PWMs offered to update the stored data after submitting filled-in credentials, despite no change occurring. Regarding phone numbers, only one PWM offered to store a phone number, while three autofilled existing phone numbers if a unique ID for the field was available. Without hints, only one PWM worked, despite the field type being phone number specific.

### 4.4. Browser interactions with PWMs over the AutoFill Framework

For our results, we report them on a per-category basis. We provide an overview of our results in Table 6.

**Basic Tests (S01-S03).** The Basic Tests worked for almost all PWMs and browsers. For Norton with native browsers, only the field currently in focus is filled, so users need to initiate multiple filling requests to complete their sign-in. In S01a, Kaspersky and Bitwarden refuse to fill fields with default HTML autocomplete attributes when they were not translated into Android predefined AutoFill hints, as Firefox or our artificial S01aa do. We find worse support for AutoSave. No PWM offered AutoSave for browsers in compatibility mode, despite this working for our stub AutoFill service in Experiment 3. This is problematic for the Microsoft Authenticator, which does not support manual addition of credentials, relying on AutoSave, CSV imports or synchronization with a Microsoft account. Norton and Keepass2Android did not work for any of the three categories. In cases where AutoSave works, no domain is attached to the account by the PWM. Therefore, all stored accounts are offered during login, defeating the purpose of automatic account selection. Furthermore, since no domain is attached, the PWM cannot show these credentials in other apps or browsers. We believe this behavior to be caused by app-specific allow lists, that are supposed to limit the domain-specific storage of account information to “trustworthy browsers”. We find the required reliance on allow lists to be a fundamental limitation and error source of PWMs across all browsers. Dashlane stores the credentials without a domain at first, and then, on a second request

TABLE 6. RESULTS OF OUR WEBSITE TESTING.

Test	Browser	cf., Table 3		Microsoft	LastPass	Keeper	Dashlane	1Password	Kaspersky	Keepass2A.	Norton	SafeInCloud	BlackBerry	Bitwarden
		# Success	# Partial											
S01 Save Basic Login	C	0	0											
	O	6	9											
	F	7	8											
S01 Fill Basic Login	C	10	10											
	O	10	11											
	F	9	9											
S01a Basic Login (Autocomplete)	O	8	9											
S01aa Basic Login (AAF Tags)	O	11	11											
S02 Username Only	C	7	7											
	O	10	10											
S02a Username Only (Autocomplete)	O	11	11											
S03 Submit Button	F	9	9											
D01 Multiple subdomains	C	10	10											
	O	11	11											
	F	9	9											
D02 Multiple Domains	C	3	7											
	O	5	9											
	F	4	7											
D03 Different Logins	C	1	9											
	O	1	10											
	F	1	9											
D04 Switch to HTTP	C	10	10											
	F	9	9											
I01 Misleading Zip-Code	C	7	7											
	O	10	10											
	F	9	9											
I02 Misleading Username	C	10	10											
	O	10	10											
	F	9	9											
I04 New Password	C	3	3											
	O	4	4											
	F	1	1											
I04aa New Password (AAF Tags)	O	7	7											
I05 No Hints	C	0	0											
	O	3	3											
I06 Textarea Field	C	8	8											
	O	8	8											
I06a Textarea Field (Autocomplete)	O	8	8											
I07 No Type	C	9	9											
	O	11	11											
I08 Maxlength	O	0	0											
	F	0	0											
AA03 Registration	C	3	3											
	O	11	11											
AA04 Lastname	C	0	8											
	O	0	9											
AA04a Last Name (Autocomplete)	O	0	8											
AA04aa Last Name (AAF Tags)	O	0	10											
AN02 Radios & Checkboxes	C	0	0											
	O	0	0											
AN05a Autocomplete Off	C	2	3											
	O	8	8											
	F	9	9											
AN06a Multiple Fields	C	0	1											
	O	0	4											
AN06aa Multiple Fields (AAF Tags)	O	0	4											
W01 Basic Auth	S	2	7											
	M	0	9											
	F	8	8											
J01 Invisible Password	O	9	9											
	F	9	9											
J03a 2-step Login	C	7	7											
	O	4	9											
J04 Multipage	C	7	7											
	O	7	7											
J06 Fake Password	C	0	3											
	O	1	11											
	F	7	8											

with the same domain, asks whether the credentials should be connected to that domain instead of the browser. When a domain only asks for the username (S02), the AAF for native browsers will send a correct request. Some mobile PWMs solve these interactions, while previous work has shown desktop PWMs to be unable to do so [1]. The success rate is worse for compatibility mode browsers, where Android tries to "guess" the presence of login fields.

**Domain Related Tests (D01-D04).** Using the same login credentials across different subdomains (D01), some PWMs explicitly ask for confirmation before continuing. The opposite case, i.e., using different credentials for different subdomains (D03), is problematic in all PWMs except Keepass2Android. While most PWMs display all credentials on all subdomains, they sometimes only list the usernames, making a selection of the correct credential difficult. Microsoft Authenticator and SafeInCloud will overwrite existing credentials between subdomains without asking. With HTTP instead of HTTPS (D04), Firefox and compatibility browsers triggered AutoFill requests, potentially sending unencrypted credentials.

**Input Field Tests (I01-I08).** The first interaction in this category included non-authentication input fields meant for ZIP codes (I01). Within compatibility mode, three PWMs wrongly insert the password into this field, leaking the password in plain text. Only Kaspersky does this for native browsers. We retested all other PWMs and confirmed that it was not a browser-specific problem. A mislabeled username field (I02) works for all tested interactions except Dashlane with native browsers. If a password change form contains new password fields (I04), even when labeled with the correct AutoFill attribute "new-password" (I04a), almost all PWMs filled it with the old password. This can be explained by our results from Experiment 3, where Firefox returns "password" instead of "newPassword" in the AutoFill request for "new-password" autocomplete HTML attributes. Browsers in compatibility mode return no values for this field, making it look like a standard password field. If there are no hints or attributes about the type of field at all (I05), most PWMs fail due to a lack of information. A textarea as input field for username (I06) is usually treated as a regular field, working with most PWMs and browser combinations. Firefox does not even send a request for this and the following tests. When using an autocomplete attribute (I06a), two more PWMs succeed. A password field without a type (I07) works for all PWM and browser combinations but leaks the credentials on screen. The max-length attribute on a field (I08) is being ignored by all PWMs, failing this interaction.

**Additional Input Field Tests (AA03-AN06aa).** First, we discuss the additional fields used for registration (AA03). For compatibility with browsers and Firefox, additional fields are filled out accidentally. This can be explained similarly to I04 where missing or wrong information led PWMs to fill old instead of new passwords. Despite a lot of the PWMs supporting additional fields like "last name", filling out a form with a "last name" field (AA04) was impossible for all of them. Two entered the username into

the “last name” field. This does not change with additional autocomplete attributes (AA04a). Android AutoFill-specific autocomplete fields (AA04aa) prevent one PWM from accidentally filling the “last name” field with the username. Radio buttons and checkboxes (AN02) were ignored by all PWMs. Next is a website containing autocomplete attributes and one field with `autocomplete="off"`, which explicitly disables AutoFill (AN05a). Compatibility mode browsers tried to fill this field despite the attribute, failing this test. Finally, filling fields for name, address, phone number, and credit cards (AN06a) were not fully supported by any PWM despite support for these fields being available in the AAF. In compatibility mode, only Microsoft Authenticator can fill in the phone number. It excludes the country code, except for the second digit (e.g., only the 9 of +49 is included), leading to an always wrong phone number for country codes with more than one digit.

**Web Standards & JavaScript Tests (W01-J06).** Basic Auth (W01) is an exception. It generates input fields as native browser UI elements and not as HTML. The Samsung Internet Browser triggers an AutoFill request for five PWMs that failed the other tests. The two PWMs, which work with this browser, also fill this request. However, the resulting login data is not connected to domains but to the app. Testing with our logging AutoFill service results in no domain being transferred. This indicated that the browser seems to perform its own PWMs whitelisting. Opera Mini does not transfer domains for Basic Auth requests to any PWMs. Regarding JavaScript, when hiding the password field until key input into the username field is detected (J01), PWMs generally fills the password as well. This becomes impossible for browsers in compatibility mode. In two cases for native browsers, the password field was visible but not filled. However, no further AutoFill request can be triggered when clicking the new field, so this interaction fails. For the multistep login on the same page and across two pages (J03a & J04), we excluded Firefox because, in our initial test, it did not trigger an AutoFill request for username-only forms (S02). In a login spanning multiple pages (J04), three PWMs with compatibility mode browsers and four with native browsers, it failed because after submitting the username, the AutoFill request cannot be started again on the second page with the password field. A dummy password field that is being replaced by a normal password field upon interaction (J06) is problematic for almost all combinations. For the native group, tapping the dummy field multiple times works.

## 5. Limitations

Our work has limitations common to large-scale Android security research, static analysis, and qualitative studies. We downloaded free Android apps from Google Play (EEA country location-based); therefore, our app sample is subject to bias due to e.g., geo-blocking or exclusion of paid apps. To combat this, we followed the best-effort approach to collect as many free apps as possible and aligned our apps

with the AndroZoo [23] set of apps. Furthermore, free apps make up around a vast majority of 97% of Google Play apps in 2023 [46]. This leaves our dataset in line with the previous Android security research [47]–[59], and we are confident that it is generalizable to the Android ecosystem. Our static analysis approach may have missed API calls, such as obfuscation using reflection or sophisticated forms of code loading. However, previous work found that those are not typically used for default obfuscation approaches for Android [28]. Hence, we believe that the 639,731 APKs in our analysis represent up-to-date and relevant examples of using authentication APIs in free Android apps from Google Play. In Section 4, we examined the top 100 apps and, within those, the most critical browsers on the Google Play Store, the most popular Android Store. We consider to have captured the most relevant interaction issues for a large base of users by market share [31], [60], [61]. Identifying the causes and general applicability of the inconsistencies we encountered was challenging in some cases. However, our rigorous testing across a large domain of browsers, apps, and PWMs illustrates how edge cases caused issues within the AAF. Therefore, we are confident that identified issues concern all apps and are not limited to our set of apps.

## 6. Discussion

We discuss our findings, connect them to our initial research questions, and provide an outlook on solutions.

**RQ1: How many apps use each authentication API.** The Credentials API still suffers from a low early adoption rate, which we expect to improve over time (cf., Section 3.3). The AAF, on the other hand, is enabled across all apps. In most cases, the AAF attributes were used to disable AutoFill features in apps (cf., Section 4). To address this, Android could provide attributes that enable developers to specify login forms, automatically disabling AutoFill for other views and reducing annoyance for developers. We also found low and erroneous use of AutoFill hints, despite Android Studio [62] providing linter warnings for these hints.

**RQ2: What limitations do PWMs encounter when using the AAF.** Our findings suggest multiple challenges with the AAF. To future-proof our results against the Credentials API, we also provide an estimation of how these issues translate to this new API. Most notably, from Google Chrome to Firebase, first-party and popular apps and libraries had no or limited support for the AAF, leading to subpar experiences for third-party PWMs. Google’s stance on this seems to change with the recent Google Chrome Beta [63], but there is no indication that this change will transfer to all their services and products. Since most mobile browsers are based on the Chromium browser, the missing support for the AAF causes many challenges in the browser to PWM interaction (cf., Section 4.2). Support for a translation layer from autocomplete attributes to AutoFill hints could go a long way, as our tests showed for Firefox and many of our artificial “-aa” use cases. These browsers serve

as the base for WebViews, so this would also contribute to improving support in all WebView apps. However, the situation on the web is still not good, and rather than suggested the recommendations of a unified framework or standard to enable more direct support between PWMs and browsers by Huaman *et al.*, the AAF as such a framework could not solve all of these issues. Support for actions like password change (I04) is bad even in browsers like Firefox that tend to support a few more cases, and the Credentials API does not currently support a workflow for password changes at all, so these interactions will likely continue to fail for it. Finally, the manual blocking and allowance approach of the compatibility framework leads to overburdened usability challenges between individual browsers and PWMs (cf., Section 3.5), and should be addressed through a general approach to declaring browsers or AutoFill services (PWMs) in the ecosystem, e.g., through user choice for apps to enable an app as browser, that can request passwords for multiple domains, should be introduced for both the Credentials API and the AAF. Finally, support for additional fields like credit card information and addresses, which are supported in many PWMs [64]–[66] on desktop and have become a staple of browser autofilling [67], [68], is currently bad both in apps (K0-T3) and the web (AN02-AN06). The Credentials API currently provides no support for these additional fields despite being fit to handle information with high security requirements like credit card information, making these cases unsupported in both frameworks.

**RQ3: How can stakeholders of the Android ecosystem better support the interaction between PWMs and apps.** Login-specific factors like AutoFill hints could, in straightforward cases, be implicitly derived, e.g., for password fields or credit card information. With AutoFill hints implicitly derived this way, they could be mandatory for the remaining fields, and an option to disable them for all fields of unrelated forms. This would cause a minimal burden for developers while clearing any ambiguities and errors the AAF may cause. The New Credentials API allows developers to handle credentials directly, skipping user input. A lot of the errors happening in our AutoFill testing (cf., Section 4.2) could be fixed outright using such an approach. However, as we have seen in the limitation section, support for many of the interactions that fail for the AAF are not supported in the Credentials API either, leaving to no improvement for users of this API despite the added effort put on developers when they integrate the API. We suggest that the Credentials API could cooperate more strongly with the AAF and existing structures like AutoFill hints, to encourage a higher adoption rate and limit implementation effort. Furthermore, we suggest the API serves as a communication layer for all high-security information on Android, including areas like credit card information. Finally, our results demonstrate that on the Web, effort even beyond that of browsers and APIs is required, a standard for form filling on websites is still missing, in line with the findings of Huaman *et al.*

## 7. Conclusion

In this work, we investigated the interaction of Android apps and PWMs to better understand their impact on mobile authentication security and usability. We performed a mixed-methods analysis of Android authentication APIs to measure their use in the ecosystem and gain insights into their impact on authentication security. We identified a lack of adoption and a challenging level of complexity for developers, with many edge cases around the AAF, causing security and usability issues in the interaction of Android apps and PWMs. We recommend that both authentication APIs address the challenges we identified and work towards improving the usability for app developers and users.

## References

- [1] N. Huaman, S. Amft, M. Oltrogge, Y. Acar, and S. Fahl, “They Would do Better if They Worked Together: The Case of Interaction Problems Between Password Managers and Websites,” in *2021 IEEE Symposium on Security and Privacy (SP)*, Mar. 2021.
- [2] S. Zhang, S. Pearman, L. Bauer, and N. Christin, “Why people (don’t) use password managers effectively,” in *Fifteenth Symposium on Usable Privacy and Security, SOUPS 2019, Santa Clara, CA, USA, August 11-13, 2019*, H. R. Lipford, Ed., USENIX Association, 2019. [Online]. Available: <https://www.usenix.org/conference/soups2019/presentation/pearman>.
- [3] Android Open Source Project, *Optimize your app for autofill* | *Android Developers*, <https://developer.android.com/guide/topics/text/autofill-optimize> (visited on 01/15/2023).
- [4] Android Open Source Project, *Android Developers Blog: Getting your Android app ready for Autofill*, <https://android-developers.googleblog.com/2017/11/getting-your-android-app-ready-for.html> (visited on 01/15/2023).
- [5] Android Open Source Project, *credentials* | *Jetpack* | *Android Developers*, <https://developer.android.com/jetpack/androidx/releases/credentials> (visited on 06/29/2023).
- [6] N. Huaman, *Passwords to go*, Sep. 2024. [Online]. Available: [osf.io/2b5qh](https://osf.io/2b5qh).
- [7] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, “Password managers: Attacks and defenses,” in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA: USENIX Association, Aug. 2014, pp. 449–464. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver>.
- [8] S. Aonzo, A. Merlo, G. Tavella, and Y. Fratantonio, “Phishing attacks on modern android,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM, 2018, pp. 1788–1801. [Online]. Available: <https://doi.org/10.1145/3243734.3243778>.
- [9] J. Lee, A. Chen, and D. S. Wallach, “Total recall: Persistence of passwords in android,” in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, The Internet Society, 2019. [Online]. Available: <https://www.ndss-symposium.org/ndss-paper/total-recall-persistence-of-passwords-in-android/>.
- [10] X. Lin, P. Ilija, and J. Polakis, “Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 507–519.
- [11] S. Oesch and S. Ruoti, “That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers,” in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds., USENIX Association, 2020, pp. 2165–2182. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/oesch>.

- [12] S. Oesch, A. Gautam, and S. Ruoti, "The emperor's new autofill framework: A security analysis of autofill on ios and android," in *ACSAC '21: Annual Computer Security Applications Conference, Virtual Event, USA, December 6 - 10, 2021*, ACM, 2021, pp. 996–1010. [Online]. Available: <https://doi.org/10.1145/3485882.3485884>.
- [13] Z. Li, W. He, D. Akhawe, and D. Song, "The Emperor's new password manager: Security analysis of web-based password managers," in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA: USENIX Association, Aug. 2014, pp. 465–479. [Online]. Available: [https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/li\\_zhiwei](https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/li_zhiwei).
- [14] S. Chiasson, P. C. van Oorschot, and R. Biddle, "A usability study and critique of two password managers," in *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4, 2006*, A. D. Keromytis, Ed., USENIX Association, 2006. [Online]. Available: <https://www.usenix.org/conference/15th-usenix-security-symposium/usability-study-and-critique-two-password-managers>.
- [15] S. G. Lyastani, M. Schilling, S. Fahl, M. Backes, and S. Bugiel, "Better managed than memorized? studying the impact of managers on password strength and reuse," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18, Baltimore, MD, USA: USENIX Association, 2018, pp. 203–220.
- [16] S. Seiler-Hwang, P. Arias-Cabarcos, A. Marín, F. Almenares, D. Díaz-Sánchez, and C. Becker, "I don't see why I would ever want to use it" Analyzing the Usability of Popular Smartphone Password Managers," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1937–1953.
- [17] H. Ray, F. Wolf, R. Kuber, and A. J. Aviv, "Why older adults (don't) use password managers," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds., USENIX Association, 2021, pp. 73–90. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/ray>.
- [18] S. Oesch, S. Ruoti, J. Simmons, and A. Gautam, "It Basically Started Using Me:" An Observational Study of Password Manager Usage," in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–23.
- [19] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv, "Why users (don't) use password managers at a large educational institution," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, 2022, pp. 1849–1866. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity22/presentation/mayer>.
- [20] S. Zibaei, D. R. Malapaya, B. Mercier, A. Salehi-Abari, and J. Thorpe, "Do password managers nudge secure (random) passwords?" In *Eighteenth Symposium on Usable Privacy and Security, SOUPS 2022, Boston, MA, USA, August 7-9, 2022*, S. Chiasson and A. Kapadia, Eds., USENIX Association, 2022, pp. 581–597. [Online]. Available: <https://www.usenix.org/conference/soups2022/presentation/zibaei>.
- [21] S. Zibaei, A. Salehi-Abari, and J. Thorpe, "Dissecting nudges in password managers: Simple defaults are powerful," in *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 211–225. [Online]. Available: <https://www.usenix.org/conference/soups2023/presentation/zibaei>.
- [22] S. Amft, S. Höltervennhoff, N. Huaman, Y. Acar, and S. Fahl, "would you give the same priority to the bank and a game? i do Not!" exploring credential management strategies and obstacles during password manager setup," in *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 171–190. [Online]. Available: <https://www.usenix.org/conference/soups2023/presentation/amft>.
- [23] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16, Austin, Texas: ACM, 2016, pp. 468–471. [Online]. Available: <http://doi.acm.org/10.1145/2901739.2903508>.
- [24] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity22/presentation/arp>.
- [25] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1332–1349.
- [26] A. Desnos, "Androguard: Reverse engineering, Malware and goodware analysis of Android applications ... and more (ninja !)" (), [Online]. Available: <https://github.com/androguard/androguard> (visited on 11/02/2024).
- [27] Android Open Source Project, "String resources." (), [Online]. Available: <https://developer.android.com/guide/topics/resources/string-resource> (visited on 09/30/2024).
- [28] D. Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, "A Large Scale Investigation of Obfuscation Use in Google Play," in *Proc. 34th Annual Computer Security Applications Conference (ACSAC'18)*, ACM, 2018.
- [29] Android Open Source Project, *View : setAutofillHints | Android Developers*. [Online]. Available: [https://developer.android.com/reference/android/view/View.html#setAutofillHints\(java.lang.String..\)](https://developer.android.com/reference/android/view/View.html#setAutofillHints(java.lang.String..)) (visited on 01/15/2023).
- [30] Android Open Source Project, *View : setImportantForAutofill | Android Developers*. [Online]. Available: [https://developer.android.com/reference/android/view/View.html#setImportantForAutofill\(int\)](https://developer.android.com/reference/android/view/View.html#setImportantForAutofill(int)) (visited on 01/15/2023).
- [31] AndroidRank, *Free Android Market Data, App and Developer History | since 2011*, <https://androidrank.org/> (visited on 01/15/2023).
- [32] Android Open Source Project, "Debug your app | Android Developers." (), [Online]. Available: <https://developer.android.com/studio/debug> (visited on 09/22/2020).
- [33] T. Hamilton, *UIAutomatorViewer Tutorial: Inspector for Android Testing*, <https://www.guru99.com/uiautomatorviewer-tutorial.html> (visited on 01/15/2023).
- [34] Android Open Source Project, *R.styleable : importantForAutofill | Android Developers*. [Online]. Available: [https://developer.android.com/reference/android/R.styleable#View\\_importantForAutofill](https://developer.android.com/reference/android/R.styleable#View_importantForAutofill) (visited on 01/15/2023).
- [35] Android Open Source Project, "R.styleable : autofillHints | Android Developers." (), [Online]. Available: [https://developer.android.com/reference/android/R.styleable#View\\_autofillHints](https://developer.android.com/reference/android/R.styleable#View_autofillHints) (visited on 01/15/2023).
- [36] Google, *com.google.android.gms.auth.api.credentials | Google Play services | Google Developers*. [Online]. Available: <https://developers.google.com/android/reference/com/google/android/gms/auth/api/credentials/package-summary#targeting-android-and-above> (visited on 01/15/2023).
- [37] Google, "Easily add sign-in to your Android app with FirebaseUI." (), [Online]. Available: <https://firebase.google.com/docs/auth/android/firebaseui> (visited on 01/15/2023).
- [38] Google, *GitHub - firebase/FirebaseUI-Android: Optimized UI components for Firebase*, <https://github.com/firebase/firebaseui-android> (visited on 01/15/2023).
- [39] Google, *com.google.android.gms.auth.api.credentials | Google Play services | Google Developers*. [Online]. Available: <https://developers.google.com/android/reference/com/google/android/gms/auth/api/credentials/package-summary> (visited on 01/15/2023).
- [40] Google, *Add full support for Android O and update deps by SUPER-CILEX · Pull Request #850 · firebase/FirebaseUI-Android · GitHub*, <https://github.com/firebase/FirebaseUI-Android/pull/850> (visited on 01/15/2023).
- [41] M. Bai, "[webview autofill] implement local prediction (i7e832aa1)." (), [Online]. Available: <https://chromium-review.googlesource.com/c/chromium/src/+1174977> (visited on 01/17/2022).

- [42] Google. "Obtain an allowlist of privileged apps." (), [Online]. Available: <https://developer.android.com/identity/sign-in/credential-provider%5C#obtain-allowlist> (visited on 05/29/2024).
- [43] Google, *Permissions and APIs that Access Sensitive Information - Play Console Help*, <https://support.google.com/googleplay/android-developer/answer/9888170> (visited on 01/15/2023).
- [44] Genymobile and R. Vimont, *GitHub - Genymobile/scrcpy: Display and control your Android device*, <https://github.com/Genymobile/scrcpy> (visited on 01/15/2023).
- [45] W3C, *HTML Standard*. [Online]. Available: <https://html.spec.whatwg.org/multipage/form-control-infrastructure.html#attr-fe-autocomplete> (visited on 11/29/2022).
- [46] 42matters AG. "What is the distribution of free versus paid on google play? - google play statistics and trends 2023." (), [Online]. Available: <https://42matters.com/google-play-statistics-and-trends#free-vs-paid-distribution> (visited on 06/27/2023).
- [47] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones," 2010.
- [48] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*, ACM, 2013.
- [49] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Proc. 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*, ACM, 2011.
- [50] W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," in *Proc. 20th Usenix Security Symposium (SEC'11)*, USENIX Association, 2011.
- [51] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," in *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*, ACM, 2012.
- [52] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL Development in an Appified World," in *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*, ACM, 2013.
- [53] V. Tendulkar and W. Enck, "An Application Package Configuration Approach to Mitigating Android SSL Vulnerabilities," in *Proceedings of the IEEE Mobile Security Technologies workshop (MoST)*, IEEE, 2014.
- [54] M. Conti, N. Dragoni, and S. Gottardo, "MITHYS: Mind the Hand You Shake—Protecting Mobile Devices from SSL Usage Vulnerabilities," in *Security and Trust Management*, Springer, 2013, pp. 65–81.
- [55] S. Fahl, M. Harbach, M. Oltrogge, T. Muders, and M. Smith, "Hey, you, get off of my clipboard - On How Usability Trumps Security in Android Password Managers," in *Proc. 17th International Conference on Financial Cryptography and Data Security (FC'13)*, Springer, 2013.
- [56] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To Pin or Not to Pin—Helping App Developers Bullet Proof Their TLS Connections," in *Proc. 24th Usenix Security Symposium (SEC'15)*, USENIX Association, 2015.
- [57] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security," in *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)*, IEEE, 2017.
- [58] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators," in *Proc. 39th IEEE Symposium on Security and Privacy (SP'18)*, IEEE, 2018.
- [59] M. Oltrogge, N. Huaman, S. Amft, Y. Acar, M. Backes, and S. Fahl, "Why eve and mallory still love android: Revisiting tls (in)security in android applications," in *In 30th USENIX Security Symposium, USENIX Security '21, Vancouver, B.C., Canada, August 11-13, 2021*, USENIX Association, Aug. 2021. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/oltrogge>.
- [60] M. Iqbal. "App download and usage statistics (2021)." (2021), [Online]. Available: <https://www.businessofapps.com/data/app-statistics/> (visited on 12/28/2021).
- [61] J. Anthony. "Number of apps in leading app stores in 2021/2022: Demographics, facts, and predictions." (), [Online]. Available: <https://financesonline.com/number-of-apps-in-leading-app-stores/> (visited on 12/28/2021).
- [62] Android Open Source Project, *Android Lint Checks - Android Studio Project Site*, <https://developer.android.com/studio/write/lint> (visited on 10/17/2023).
- [63] Leopeva64. "Leopeva64 on X: In Chrome for Android there is a new section called "Autofill Options" where you can choose whether to use Chrome or "other providers" to save and fill passwords, passkeys, payment methods, and addresses." (), [Online]. Available: <https://x.com/Leopeva64/status/1723828148939210948> (visited on 05/29/2024).
- [64] LastPass US LP. "Make Life Easier with Save and Autofill." (), [Online]. Available: <https://www.lastpass.eu/features/autofill> (visited on 05/29/2024).
- [65] Bitwarden, Inc. "Auto-fill Cards & Identities." (), [Online]. Available: <https://bitwarden.com/help/auto-fill-card-id/> (visited on 05/29/2024).
- [66] Dashlane, Inc. "Manage payment cards and accounts saved in Dashlane." (), [Online]. Available: <https://support.dashlane.com/hc/en-us/articles/202625362-Manage-payment-cards-and-accounts-saved-in-Dashlane> (visited on 05/29/2024).
- [67] Google. "Manage your Google payment info." (), [Online]. Available: <https://support.google.com/accounts/answer/9244912?hl=en> (visited on 05/29/2024).
- [68] M. Rodaro, Mozinet, M. Ghelman, ErlingR, and Fabi. "Automatically fill in credit card information on Firefox for Android." (), [Online]. Available: <https://support.mozilla.org/en-US/kb/automatically-fill-credit-card-information-firefox> (visited on 05/29/2024).
- [69] Google, *Use of the AccessibilityService API - Play Console Help*, <https://support.google.com/googleplay/android-developer/answer/10964491?hl=en> (visited on 01/15/2023).
- [70] Android Open Source Project. "HintConstants | Android Developers." (), [Online]. Available: <https://developer.android.com/reference/androidx/autofill/HintConstants> (visited on 01/15/2023).
- [71] Android Open Source Project, *Build autofill services | Android Developers*. [Online]. Available: <https://developer.android.com/guide/topics/text/autofill-services#advanced> (visited on 01/15/2023).
- [72] Google. "Smart Lock." (), [Online]. Available: <https://developers.google.com/identity/smartlock-passwords/android/overview?hl=en> (visited on 01/15/2023).
- [73] Google, *Start Integrating Google Sign-In into Your Android App*, <https://developers.google.com/identity/sign-in/android/start-integrating?hl=en> (visited on 02/07/2023).
- [74] Google. "com.google.android.gms.auth.api.identity | Google Play services | Google Developers." (), [Online]. Available: <https://developers.google.com/android/reference/com/google/android/gms/auth/api/identity/package-summary> (visited on 01/15/2023).
- [75] Android Open Source Project. "Integrate Credential Manager with your credential provider solution." (), [Online]. Available: <https://developer.android.com/identity/sign-in/credential-provider> (visited on 05/28/2024).

## Appendix A. Authentication on Android

Below, we provide background information for authentication APIs on Android. Figure 2 gives an abstract overview of the general authentication flow.

**Android AutoFill Framework and the Compatibility Framework.** The integrated and default framework for automated authentication on Android is the AAF [3], [4]. It was introduced in 2017 to provide a specified API for PWMs

and similar services to interact with applications [4]. This was done to replace the previous workaround for PWMs using the Accessibility API, which to enable accessibility support allows circumventing Android apps’ isolation, access files, screens, and inputs, and therefore, since 2021, requires special review by Google to be approved for Google Play [69]. The AAF, on the other hand, offers an interface for views to request passwords from an AutoFill service, keeping isolation intact and reducing capabilities. The AutoFill service obtains information required to identify the app and the credentials it requests through the API. Then, one PWMs set up as the default AutoFill service can decide whether to provide which credentials. Support for this framework is integrated into the default input fields within the AAF and developers may mark fields as not relevant for AutoFill [34], and can provide hints on the type of information requested in the app’s input fields [35], [70].

Browsers and apps that do not use Android native Views have to manually implement interfaces related to the AAF. To allow for support in older apps like browsers that do not use Android-native Views to render websites, a compatibility mode [71] was introduced which internally uses the accessibility framework similarly to the previous mechanism. To use this mode, however, PWMs needs to allow listing the apps they use it for, and the PWM itself needs to be whitelisted by the system. In our diagram, the System provides the AAF API including the mentioned options to apps, and then allows PWM to register as AutoFill Services that the user can choose.

**Credentials API.** The Credential Manager API and Credential Provider Interface, which we shorten to Credentials API, combine to a new API introduced in Android 14 [5]. From a user perspective, it works similarly to the AAF, but using a more streamlined interface.

This API combines several previous authentication-related APIs, among them Smart Lock [72], “sign-in with Google” [73] and One Tap API [74]. It provides support for credential providers, which can refer to PWM, but also supports passkey and federal workflows and may in the future be extended with other approaches. [75]. From a developer perspective, it requires a higher workload than the AAF since developers need to manually integrate it into their apps, select and handle the types of credentials they support. However, this kind of deep integration prevents a few of the pitfalls we discover in the AAF in this paper. From a high-level overview, it works similar to the AAF, but the API is the Credential Manager framework [5], and the PWM is the credential provider. The user can choose the default credential provider, and alternatives whenever he tries to log in.

## Appendix B. AutoFill Framework Analysis—Additional Insights

Due to the broad number of experiments we conducted and insights we found, not all could be fit in the paper.

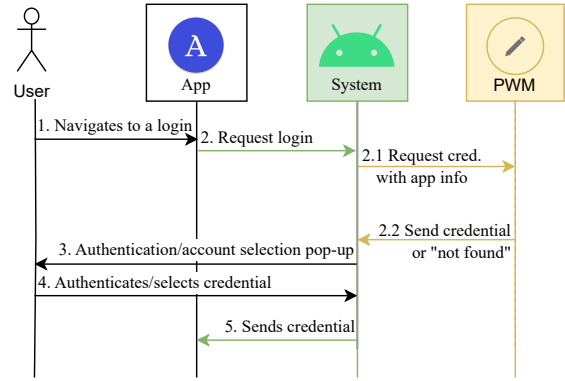


Figure 2. Overview over how API requests work for the Authentication APIs

TABLE 7. SEARCH RESULTS FOR PWMs BY POPULARITY. FULL APP NAMES AND LINKS IN REPLICATION PACKAGE [6]

	Autofill Framework	Downloads (in Millions)
<b>Password Manager</b>		
Microsoft Authenticator	●	50
Keeper Password Manager	●	10
LastPass Password Manager	●	10
Wi-Fi password manager		10
1Password Password Manager	●	1
Bitwarden Password Manager	●	1
BlackBerry Password Keeper	●	1
Dashlane Password Manager	●	1
(Kaspersky) Password Manager	●	1
Keepass2Android Password Safe	●	1
Norton Password Manager	●	1
Password Manager SafeInCloud	●	1
True Key™ by McAfee		1
KeePassDroid		1
aWallet Password Manager		1
PassKeep: Password Manager		1
My Passwords - Password Manager		1
Password Safe - Secure Password Manager		1
Password Saver		1

Additional insights into how we constructed Interactions and how interactions worked with our setup can be found here.

### B.1. Constructing Interactions

We constructed multiple interactions, in part based on the three experiments we conducted (Experiment 1, 2 and 4) and on the interactions already provided by Human *et al.* [1]. In the following, we will provide a quick overview of which interactions we adapted, which ones we added and why we added them.

**App interactions.** Apps using normal Android views fully support the AAF by default, since it is a platform feature integrated into the views. Our initial app tests therefore focus on the level of support in edge cases. Since the

TABLE 8. SELECTION OF OUR BROWSERS.

Browser (short)	Version	Mio. Installations	Comp.-Whitelist	AutoFill Support	AutoFill Hints	Additional Info
Google Chrome	95.0.4638.74	5,000	●	C	-	-
Samsung Internet Browser	15.0.6.3	1,000	●	C	-	-
UC Browser	13.4.0.1306	500	○	-	-	-
Opera Mini	61.0.2254.59937	500	●	N	○	●
Firefox Browser	94.1.2	100	●	N	●	-
Yandex Browser	21.11.1.114	100	○	-	-	-
Opera Browser	65.2.3381.61420	100	●	-	-	-
Mi Browser Pro	12.15.2-gn	100	○	N	○	●
Phoenix Browser	8.8.3.3440	100	○	N	○	●
UC Mini	12.12.10.1227	100	○	-	-	-
Dolphin Browser	12.2.9	50	○	N	○	●
UC Browser Turbo	1.10.9.900	50	○	-	-	-
Puffin Web Browser	9.4.1.51004	50	○	-	-	-
Brave Private Browser	1.31.91	50	●	C	-	-
Aloha Browser	3.9.3	10	○	-	-	-
Via Browser	4.3.2	10	○	N	○	●
Hi Browser	2.1.090	10	○	N	○	●
APUS Browser	3.1.5	10	○	N	○	●
Ecosia	4.4.1	10	○	-	-	-
Tor Browser	10.5.10	10	○	N	●	-
Web Browser & Fast Explorer	5.7.1	10	○	N	○	●
Kiwi Browser	95.0.4638.75	10	○	N	○	●
Opera browser beta	66.0.3425.61578	10	●	-	-	-
Web Browser & Explorer	3.7.0	10	○	N	○	●
Microsoft Edge	95.0.1020.55	10	●	C	-	-
Opera Touch	2.9.6	10	○	N	○	●
Maxthon browser	6.0.1.4800	10	○	N	○	●
DuckDuckGo Privacy Browser	5.102.2	10	○	N	○	●

N: Native, C: Compatibility mode, - unsupported.  Selected browser,

AAF provides the unique App Identifier, domain cases from Huaman *et al.* where the issue is to identify a service don't apply here at all. Furthermore, since the AAF will provide all input fields it finds that have not been explicitly excluded, cases from Huaman *et al.* like JavaScript and basic login don't apply either, since obscuring login fields to the point where the AAF, and therefore Android as a whole, doesn't recognize them will cause other bugs within the Android Ecosystem and be considered a problem of the App itself. We therefore focus on how much information login fields provide regarding AutoFill hints, IDs, input field types and field description (L1-L6) to find out how PWMs react and what level of information the different PWMs need. We also test WebViews as half browser, half native app implementations (L7 & L8) to identify how well they work. Furthermore, multi-view or split logins (P1-P3, N1-N3, Z1 & Z2) require context-awareness from the PWMs to fill the correct input on the correct fields. Finally, we test the support for additional information: credit card info (K1-2) and phone numbers (T1-T3). The variants of each letter of our cases (e.g., L1, L2, L3, L4) indicate the different level of hints we provide. As visible in Table 9, we also included interactions to test autosave of PWMs, encompassing all test cases with a 0 (L0, Z0, P0...) and Z1. Both Z interactions

TABLE 9. SCENARIOS FOR OUR APPLICATION TO PWM COMPARISON IN EXPERIMENT 2.

Test	Goal	Description	Correct Type	AutoFill Hints	Identifiable ID	Label
L0	Save	Username and Password	●	●	●	●
L1	Fill	Username and Password	●	●	●	●
L2	Fill	Username and Password	●		●	●
L3	Fill	Username and Password	●			●
L4	Fill	Username and Password	●			
L5	Fill	Username and Password (Type Text)		●	●	●
L6	Fill	Username and Password (Type Text)			●	●
L7	Fill	Login in WebView (local File)	*	*	*	*
L8	Fill	Login in WebView (external Website)	*	*	*	*
P0	Save	Single Field for Password	●	●	●	●
P1	Fill	Single Field for Password	●	●	●	●
P2	Fill	Single Field for Password	●		●	●
P3	Fill	Single Field for Password	●			●
N0	Save	Single Field for Username		●	●	●
N1	Fill	Single Field for Username		●	●	●
N2	Fill	Single Field for Username			●	●
N3	Fill	Single Field for Username				●
Z0	Save	2-step Login in separate Activities	●	●	●	●
Z1	Save	2-step Login in the same Activity	●	●	●	●
K0	Save	Fields for Credit Cards		●	●	●
K1	Fill	Fields for Credit Cards		●	●	●
K2	Fill	Fields for Credit Cards			●	●
T0	Save	Field for a Phone Number	●	●	●	●
T1	Fill	Field for a Phone Number	●	●	●	●
T2	Fill	Field for a Phone Number	●		●	●
T3	Fill	Field for a Phone Number	●			●

test the ability to autosave when the login is split across multiple activities or views, to verify that both username and password are stored despite their separation. For the other interactions, we were mainly interested in how much of the information for a field would be stored correctly, so one saving test per general type of interaction sufficed.

**Browser interactions.** Our browser interactions are more directly related to the desktop browser interactions investigated by Huaman *et al.* [1]. Here, we mainly retested interactions that reveal new details through the Autofill framework, as well as added new interactions based on the frameworks' behavior. One interesting example of interactions added is the inclusion of AutoFill attributes, since those would be used as AutoFill hints by browsers, marked by an "a" next to the test case, for example S01a. We also included a new type of interaction "aa", where we used the Android default value as autocomplete attribute instead of the usual

TABLE 10. TEST WEBSITES FOR BROWSER AND AAF.

Test	Description	Ex3	Ex5 - Comp.	Ex5 - Firefox	Ex5 - Native
<div style="background-color: #c8e6c9; width: 15px; height: 10px; display: inline-block; margin-right: 5px;"></div> New cases * W01 Doesn't fit browser categories.					
<b>Basic tests</b>					
S01	Basic page with 2 fields for username (type input) and password (type password)	●	●	●	●
S01a	S01 with added autocomplete-attributes	●			●
S01aa	S01a with default Android AutoFill hints as autocomplete-attributes				●
S02	Page with only a field for username	●	●		●
S02a	S02 with added autocomplete-attributes	●			●
S03	Button declared as input type submit	●		●	
<b>Domains</b>					
D01	Same login across multiple subdomains		●	●	●
D02	Same login across multiple domains		●	●	●
D03	Different login across multiple subdomains		●	●	●
D04	Switch to HTTP		●	●	
D06	Iframe with a page from a different domain	●			
<b>Input Fields</b>					
I01	Additional field for zip code, name and ID misleading (code)		●	●	●
I02	Misleading username		●	●	●
I04	Two additional fields to provide a new password	●	●	●	●
I04aa	I04 with Android AutoFill hints as autocomplete attributes				●
I05	Not hints, including no types regarding the fields, except for labels.	●	●		●
I06	Username field is a textarea	●	●		●
I06a	I06 with added autocomplete-attributes	●			●
I07	No type specified	●	●		
I08	password is limited in length using a maxlength-attribute	●		●	●
<b>Additional Login Fields</b>					
AA03	Additional Fields for Registration on a Page		●	●	●
AA04	An additional field for lastname, required for logging in.	●	●		●
AA04a	AA04 with added autocomplete-attributes	●			●
AA04aa	AA04a with Android AutoFill hints as autocomplete attributes				●
<b>Additional Unrelated Fields</b>					
AN02	Additional radio buttons and checkboxes.	●	●	●	●
AN05a	Additional field with autocomplete-Attribute off, which should not be filled. Autocomplete attribute for all fields provided.	●	●	●	●
AN06a	Fields for name, address, phone number and credit card information.	●	●		●
AN06aa	AA06aa with Android AutoFill hints as autocomplete attributes				●
<b>Webstandards</b>					
W01	Basic Authentication	●	*	*	*
<b>JavaScript</b>					
J01	Password field invisible until a username is entered.	●		●	●
J03a	2-step login. Password field only added after username input. Includes Autocomplete attributes.	●	●		●
J04	Username and password one after another on separate pages		●		●
J06	Fake password field replaced with real one after entering inputs.	●	●	●	●

default [45]. This is because some browsers transmitted the HTML autocomplete attribute [45] directly, which doesn't always match the default AutoFill hint [70] value, while other browsers provided a translation from autocomplete attribute to AutoFill hint. We skipped any interaction from Huaman *et al.* where, according to Experiment 1 and 3, the change they tested in the form would not result in a change of data transmitted by the AAF. As an example, W02, where multiple Authentication attempts are triggered via basic auth, resulted in an AutoFill request per basic auth request, so all PWMs behaved equally, and equal to W01, a single basic auth request, leading to no new insights in our testing. For D05 multiple redirects after a login, the prompt to autosafe is an OS level prompt, not vanishing on any number of redirects, so no additional insights would be gained. We further excluded interactions that made no sense to port over, like AN01, where multiple login-like forms are present. The AutoFill request here would be triggered for the form that the user clicked on, so additional elements on the page had no influence at all. We still checked additional fields that were part of the same form, like AA03 & AN02. A full overview of included and added interactions can be seen in Table 10.