ARTIFACT
EVALUATED
usenix
ASSOCIATION
AVAILABLE

HyTrack: Resurrectable and Persistent Tracking Across Android Apps and the Web

Malte Wessels, Simon Koch, Jan Drescher, Louis Bettels, David Klein, Martin Johns
Technische Universität Braunschweig
{malte.wessels, simon.koch, jan.drescher, louis.bettels, david.klein, m.johns}@tu-braunschweig.de

Abstract

Android apps can freely intermix native and web content using Custom Tabs and Trusted Web Activities. This blurring of the boundary between native and web, however, opens the door to HyTrack, a novel tracking technique. Custom Tabs and Trusted Web Activities have access to the default browser state to enable, e.g., seamless reuse of authentication tokens. HyTrack abuses this shared browser state to track users both in-app and across the web using the same identifier. We present several ways to hide or completely disguise the tracking from the user by integrating it into the app's UI. Depending on the used Android flavor, HyTrack leaves no visible traces at all. Furthermore, by combining basic functionalities of the Android operating system, we also show that identifiers created with HyTrack are almost impossible to get rid of. HyTrack can resurrect tracking identifiers even when users try last-resort techniques, such as changing the default browser or switching devices, making it more persistent than even evercookies were on the Web. While we do not find direct evidence that our technique is already employed, our findings indicate that all essential components are currently in place. A rapid deployment can occur at any given moment. To summarize, this paper provides an early warning of a potentially severe new tracking approach for the Android operating system that solely utilizes the intended behavior of commonly utilized Android features.

1 Introduction

HyTrack is a powerful tracking technique that has the potential to enable Trackers such as Advertisement Companies to conduct cross-app and cross-web tracking on the smartphone. Advertising companies deploy consumer tracking as a tool to improve their understanding of the user and, thus, user engagement. The primary goal of tracking is to create a faithful user profile. This allows the advertisement company to tailor services to the users, increasing sales. While this presents a powerful revenue multiplier for the company, it also has severe privacy implications. The profiles can include very personal details depending on how thorough the tracking is. A tracker that can track a user across dating or health-related sites implicitly picks up on sexual preferences, relationship status, or ailments. Consequently, privacy advocates are constantly pushing for more protection against tracking. The decision of Android to allow users to disable the device's global Google Tracking ID in 2021 was an important recent change [1] in this direction. However, tracking companies still want to keep their revenues, resulting in a tug-of-war between privacy protection and tracking.

Research on tracking vectors can be split into two main approaches, *stateful* and *stateless*. Stateful tracking leverages persistently stored identifiers, most commonly cookies, to keep track of users. The main challenges for stateful tracking are keeping identifiers persistent even if the user deletes them and ensuring that tracking across different contexts works despite the Same Origin Policy in the browser or sandboxing on Android devices. In 2010 Kamkar [2], demonstrated *evercookies*, a technique to resurrect previously deleted cookies. While this increases browser tracking capability, it does not enable cross-app tracking, as Android keeps apps sandboxed from the system and each other. Thus, apps cannot subversively exchange persistent identifiers. Android provides the solution to this, as it provides the Google Advertisement ID, a device global identifier accessible by apps. But recently, Google added the option for users to turn this identifier off, which again limits the ability for stateful cross-app tracking [1]. Stateless tracking, on the other hand, leverages device configuration, such as of the browser as initially shown by Eckersley [3] or hardware behavior as demonstrated by Das et al. [4] or Zhou et al. [5].

HyTrack adds a powerful new technique to the set of known tracking approaches that not only allows tracking of users across apps but also across websites on the same device. To achieve this, we leverage the persistent storage of the web browser that is shared across apps. Apps inadvertently use this persistent storage when they leverage advanced features such as *Custom Tabs* (CTs) or *Trusted Web Activities* (TWAs).

As multiple apps perform requests using CTs or TWAs to the same web page, they share the cookie jar, allowing the targeted page to set a persistent cookie. We show that this behavior can be achieved without the user's knowledge and even be obfuscated from the app developer through a third-party library. Thus, using third-party libraries risks exposing users to persistent cross-app tracking across all apps. Furthermore, the library can even use a backchannel to receive the tracking identifier from a Custom Tab or Trusted Web Activity and store it in the app storage, which is included in Google's backup functionality. This subsequently exposes users to persistent tracking spanning all installed browsers and even device changes.

Besides presenting HyTrack, we also conduct three large-scale studies to ascertain if our technique is already being deployed in the wild. We conduct a static analysis across our dataset consisting of the 4358 most popular Android apps to study the prerequisites for our HyTrack, i.e., usage of CT and TWA features as well as an app-specific scheme. Subsequently, we perform a no-touch traffic collection and study the traffic for any shared cookie values across different apps. Additionally, we perform a qualitative analysis, collecting traffic while interacting with apps. Finally, we run a large-scale web crawl across the Tranco [6] 500 000 and all entries on the EasyPrivacy and Exodus lists to collect indicators of collusion between known trackers and apps required for TWA usage. Additionally, we test if HyTrack can be conducted across different Android flavors and mobile browsers.

Our results show that HyTrack is possible across a wide range of configurations, and a third of popular apps already possess the required capabilities. However, we could not find any deployments of HyTrack in the wild. Therefore, there is still time to address the issue. Thus, our contributions are the following:

- We evaluate if the requirements for our tracking technique exist across Android flavors and different browsers (Section 3).
- We present HyTrack, a novel persistent cross-app and web tracking technique that can be hidden from both the user and the developer (Section 4).
- We conduct one web and two app measurements to assess how widespread the capabilities to launch such a tracking campaign are and whether there are any indicators of current deployments of HyTrack (Section 5).

2 Background

We first need to lay some groundwork to understand HyTrack and the context in which it is happening. We establish how tracking on Android and the Web works in general (Section 2.1), the interplay between Intents, Activities, and Custom Schemes (Section 2.2), and how Android interacts with and opens URLs (Section 2.3).

2.1 Tracking on Android and the Web

Tracking is an essential feature of the monetization toolkit of today's digital services. The primary objective of tracking is to re-identify users across diverse contexts, such as different websites or apps, to infer aspects of their personality. Based on the collected information, advertisement companies can build and evolve a user profile to assess their interests and match them against advertisements. This allows the serving of personalized advertisements, which increases user engagement and, subsequently, revenue.

Building a tracking profile has severe privacy implications, as the tracking companies necessarily gain personal details users did not intend to share. Simply by tracking the visited sites and services the person visits, the profile implicitly contains information on personal interests, relationships, or even health status that can be inferred. Depending on the company building this profile, the information is used directly or sold to other interested parties. In case of a data breach, completely unknown entities can take hold of this personal information.

Generally, there are two main ways to track users, either *stateless* or *stateful tracking*. Both aim to associate a unique identifier with a user, allowing the tracker to distinguish between different users.

Stateful Tracking The most common form of tracking is the stateful variant. Here, the tracker stores a unique identifier in persistent storage and attaches this identifier to subsequent actions by the tracked person. These can be events such as shopping interactions, site visits, or app usage.

On the Web, third-party cookie-based tracking is the classic example of stateful tracking. A website example.com that includes adverts from ad.com includes a JavaScript library into their website that is hosted on the third-party domain. For every request to example.com, the user's browser also sends a request to load the library from ad.com. If no tracking cookie from ad.com is set, i.e., the user has a fresh browser profile, the server of ad.com creates a new, unique identifier and stores it in a cookie. Otherwise, the request to ad.com already has the identifying cookie attached to the request, allowing ad.com to track the user across visits and across different websites, including the advertisement library in question.

This general technique, which leverages persistent storage to store a unique identifier that gets attached to all interactions with the advertiser's servers, is directly transferable to the mobile domain. If several apps share storage, they can place a unique identifier at a location accessible to other apps and use this identifier to track the user across apps. Otherwise, apps also have access to the Google Advertisement ID (gaid) – a device's global unique identifier. Using the gaid does not require the tracker to generate or store an ID themselves. They can simply use the one made available by the operating system. However, since the end of 2021, Android allows users to disable the gaid [1]. Additionally, strict sandboxing of apps greatly increases the difficulty or even makes it impossible

for apps to employ other persistent and cross-app tracking approaches. Thus, companies are turning to alternative means, such as stateless tracking [7, 8].

Stateless Tracking If the tracking company does not have the option to reidentify the user by leveraging the device's persistent state, it must derive a unique and consistent identifier by other means. This form of tracking is called *stateless* and has become increasingly common since both the Web and mobile ecosystems are making stateful tracking increasingly difficult. On Android, this is due to disabling the global gaid as well as improved sandboxing between apps, while on the Web, this is due to the phaseout of third-party cookies.

The foundation of stateless tracking is access to numerous data points that differ between users and their devices. Examples of such data points are screen dimensions or the set of available fonts. A large set of such input data points is combined to calculate a single fingerprint, which is then used as an identifier. It is also possible to leverage aspects of the underlying hardware and its drivers, as with Canvas fingerprinting on the Web [9]. Canvas fingerprinting works by rendering a specific and static image into a `canvas` tag. The resulting images differ slightly between different browsers, graphics cards, and driver versions. By computing a hash of the resulting image, aspects of the hardware that are not directly enumerable become available for fingerprinting.

While stateless tracking does not require any persistent state, it also has clear drawbacks. Fingerprinting techniques cannot always uniquely identify users, and changes to the system, such as a driver update, can change the computed identifier. This contrasts with stateful tracking, where the tracking company generates the initial values and can ensure their uniqueness and consistency for the same user.

To summarize, stateless tracking is thus less accurate but also harder to block. As it solely relies on access to benign data, no fundamental mitigations against fingerprinting are possible. For example, the screen dimensions are necessary information to ensure the app or page can render its content optimally. Equally, displaying images is a perfectly valid use case for the `canvas` tag on a website.

Some browser developers enacted measures to make their browsers more resistant to fingerprinting. The Tor browser, for example, aims to always return the same fingerprint on every device, including a fixed screen resolution [10].

2.2 Intents, Activities, and Custom Schemes

To define the UI of an Android app, the developer divides the app into individual Activities. Those Activities are not only a conceptual structure but are defined in the app's manifest XML file. They represent the containers for the UI components of an app and have a corresponding lifecycle model supported by the OS. An app needs to define at least a main Activity, representing the entry point into the app.

However, launching the app with an Activity different from the specified entry point is also possible. This functionality is implemented via Android's *Intents* mechanism. Intents are essentially messages between apps containing both an action and corresponding parameter data. An Activity registers an intent that starts the activity upon being triggered via *intent filters*. This registration can also define a custom URI scheme. Android starts the corresponding app and action when the user interacts with a URI that matches such a filter and scheme. This allows, for example, the addition of links to a website that open native apps when clicked. Even if the intent was invoked via interaction with a custom scheme URI, passing parameters is possible by encoding them into the URI.

2.3 Displaying Web Content on Android

In general, there are four ways for Android apps to open URLs and show web content to the user.

Browser Apps can construct an intent to open a URL [11]. This causes Android to select, or ask the user to select, an app that opens the URL. In most cases, this corresponds to the configured default browser. Thus, the browser is launched and moves to the foreground, moving the previous app to the background.

WebView If developers would rather not open the browser but still include web content directly into their app, they can utilize WebViews. WebViews can be directly embedded as a component into an app and render web content inside it [12]. Users can interact with the displayed web content, but a WebView is not a fully-fledged browser and lacks advanced features.

Custom Tabs Recently, Chrome on Android introduced the concept of Custom Tabs (CTs) [13] which *"gives app developers a way to add a customized browser experience directly within their app"*. Custom Tabs are instances of the default web browser rendered within the app. However, instead of being a part of the app, like WebViews, the browser manages them. They look comparable to a regular browser window, always displaying the URL bar. However, the URL bar is not interactive. The app can style it, e.g., the color can be changed. Custom Tabs share the browser state with the browser hosting the Custom Tab. This is an essential feature, as it allows displaying web content that is user session related. For example, if a user is logged into example.com in their browser and opens a Custom Tab for example.com in a Custom Tab, they are already logged in.

Trusted Web Activities If the app developer wants an even tighter integration between web content and their app, they can use Trusted Web Activities (TWAs). TWAs are a specialized version of Custom Tabs [14]. They are rendered in full-screen mode without displaying the URL bar. Consequently, TWAs can be seamlessly integrated into existing apps. However, due to their tight integration into the app, they rely on the displayed website and the app having a mutual trust relationship.

This trust relationship is implemented in the form of a Digital Asset Link (DAL) between the website and the app [15, 16] to prevent misuse. If a TWA is navigated to a website without a DAL, it is downgraded to a Custom Tab.

A DAL establishes the mutual trust relationship between a website and a mobile app. For the DAL to work, the website must host a specific JSON linking it to the app, named `assetlinks.json`. The app has to contain a corresponding entry in its manifest that links it to the website. Android checks both references before opening the TWA to ensure that both are linked via a DAL. Refer to Appendix B for an example of a DAL.

Besides Trusted Web Activities, another use case for DALs is apps registering the HTTP scheme. If an app registers an HTTP scheme intent filter for a specific website, it signals that instead of displaying the website, the content is also available in the native app. Previously, users had the option to either open the URL in the browser or the app. But to offer this option, since Android 12, there also needs to be an association between the app and the URL in question via a DAL. In case no DAL exists, the browser is used to open the URL anyway. The app is then only opened if the user explicitly configures the app as a handler in the system settings.

Lessons Learned To summarize, Android has four ways of displaying web content: by invoking the browser, opening a WebView, a Custom Tab, or a Trusted Web Activity. Their different capabilities are summarized in Table 1. While the CT and TWA mechanisms are browser-independent, the vendor of the browser has to implement the feature. Firefox, for example, supports Custom Tabs but not Trusted Web Activities [17].

Table 1: Capabilities an app has when opening a URL in...

Capability	Browser	WebView	Custom Tab	TWA
Integration possible	○	●	●	●
Shares browser state	●[1]	○	●	●
Can hide URL bar	○	○	○	●
Can control size	○	●	◐[2]	○
Can open any URL	●	●	●	○[3]

○: unsupported, ● supported, ◐: partially supported, 1: Owns the shared state, 2: Custom Tabs can be reduced to 50 % programmatically and minimized by users, 3: Requires association via asset links.

3 Shared Storage and TWA Capabilities

Android vendors are notorious for changing and customizing the Android flavors they ship, and changes to the user interface and experience are widespread. To understand the impact of these changes on Custom Tabs and TWAs, we conducted a structured test. We evaluated if the basic functionality of CTs and TWAs, such as using the shared browser state, work as expected across common Android flavors and their deployed browsers. Additionally, we cover popular and privacy browsers.

To conduct these tests, we implemented proof of concept apps for both Custom Tabs and Trusted Web Activities and tested their functionality across devices. To identify the two apps, we assign a unique ID to both, which is passed as a GET parameter in the subsequent CT or TWA launches. Both apps open a simple website that reads a URL parameter, writes it to a cookie, and displays a list of cookie values in the HTML content and the transmitted user agent. Therefore, we can verify that the apps can use the browser's cookies as shared storage. The TWA proof of concept app additionally uses the TWA capabilities to hide the URL bar and display the website in full-screen mode, as well as registering a URL scheme to implement navigation back from the website to the app using intents (Section 2.2).

Our study encompasses six different Android flavors (Google's Android for Pixels, GrapheneOS, Samsung's OneUI, Huawei's EMUI, Xiaomi's MIUI, and OnePlus' OxygenOS) and ten different browsers: 5 vendor browsers (Chrome, Vanadium, Samsung Internet, Huawei Browser, Mi Browser), the three remaining Android browsers from the Top 5 mobile browsers [18] (Firefox, Opera, UC Browser) and two known privacy-browsers (Tor Browser, Brave). For each device, we test all preinstalled browsers. Thus, we use Chrome, which Google requires hardware vendors to bundle if they want to license the Google Play Store [19], and the vendor-specific browsers for each device, i.e., Samsung Internet, Mi Browser, Huawei Browser, and Vanadium. GrapheneOS does not license Google Play and consequently only bundles the Chromium fork Vanadium. We test the remaining browsers on the stock Google Pixel.

Setup We test both Google Stock Pixel Android 14 and GrapheneOS on a Google Pixel 6A each. We used a set of four additional devices to test the other ROMs. A OnePlus Nord (OxygenOS), a Xiaomi Poco F3 (MIUI), a Huawei P40 Lite (EMUI), and finally a Samsung Galaxy A13 (OneUI). We list the version numbers of all tested Android distributions and browsers in Table 5 in the Appendix.

Methodology For every device and browser combination, we performed the same test protocol: As a preamble for each test, we configure the default browser to the browser we want to test. We reset the browser and opened it once to conduct the initial configuration, declining all requests for permission to collect data. After this initial setup, we open the first app, which opens a Custom Tab to our example website, and observe how the Custom Tab is displayed. In particular, we watch out for notifications that the visible content is running in a browser. If the browser implements Custom Tabs correctly, it should display the browser window with the URL bar as an activity of the app. Thus, we also open the *Recents* ("Open apps") screen and verify that the CT is an activity of our app. As the app transmits a unique ID via a URL parameter, which the website then displays, we can verify that this communication channel works. If the reflected user agent

Table 2: CT and TWA Capabilities and Behavior observed during our study of ten different browsers.

Device OS (Android version) Default Browser		Google Pixel 6a Android (14) Chrome		Google Pixel 6a GrapheneOS (14) Vanadium		Samsung Galaxy A13 OneUI 5 (13) Samsung Internet		Huawei P40 Lite EMUI 12.0.0 (10) Huawei Browser		Xiaomi Poco F3 MIUI 14.0.7 (13) Mi Browser		Oneplus Nord OxygenOS 13.0 (13) Chrome	
Default Browser	CT / TWA	●	●	●	●	●†	●†	◐†	○1	◐†	○1,2	●	●
Chrome				⦸	⦸	●	●	●	●	●	●		

Device / OS Browser		Google Pixel 6A / Android (14) Opera		Firefox		Tor Browser		UC Browser		Brave	
	CT / TWA	◐†	○1	●†	○1	●†‡	○1	◐	○	●	●

●: supported, ◐: spawns new browser instead, ○: not supported, ⦸: GrapheneOS does not ship with Chrome, †: no user notification, 1: fallback to CT, 2: *no* fallback to Chrome, ‡: Tor deletes the browser state when closed, therefore it only acts as shared storage if two apps are using its CT simultaneously.

indicates that another browser than the browser under test was opened, we note down the override rule, deactivate the other browser, and restart the experiment. After closing the app, we inspect the browser's open tabs and history to determine if the Custom Tab left visible traces. Next, we examine the behavior of Trusted Web Activities using our second app. We observe if and how the TWA is displayed. Again, we verify the user agent to ensure that the chosen default browser is actually used. If the browser correctly implements TWAs, we should observe a full screen Trusted Web Activity without the URL bar. We verify that the TWA is displayed as an activity of our app in the Recents screen, and again also watch out for notifications informing the user of the context switch. The second app also transmits a unique ID via a URL parameter, so if the browser's state is shared between different apps, we should now observe the cookie values set by both apps. Subsequently, we navigate back to the app using the hyperlink to a registered scheme and verify that this return navigation works as expected. After closing the app, we search for traces of the TWA usage in the open browser tabs and browser history. Last, we manually open the browser and navigate to our website to check if the cookies set by the CT and TWA are also available in the regular browser context.

Results Table 2 provides an overview of the Custom Tab and Trusted Web Activity capabilities that we observed for the different browsers. At the time of our experiments, only Chrome, Vanadium, Samsung Internet, Brave, Firefox, and Tor Brower correctly implemented CTs. Opera, Huawei Browser, and Mi Browser spawned a new browser instead. They lacked the seamless user experience of CTs. Only Chrome, Samsung Internet, Vanadium, and Brave supported TWAs.

We observed interesting fallback behavior when our app requested a TWA, but the configured default browser does not support them. On all devices but the Xiaomi Poco F3, the system would fall back to Chrome to create the TWA. The Xiaomi Poco F3 strictly favored the Mi Browser if installed, even when Chrome was set as the default browser. When we deactivated the Chrome browser to force the system to use our desired default browser, the browsers without TWA support would fall back to their CT behavior.

Only Chrome and Vanadium inform the user that a context switch to the browser was happening. Chrome informs the user of a CT context switch via a toast message (small disappearing pop-up) and of TWA context switches via a snack bar [20] message (small pop-up bar at the bottom of the screen). For the Huawei Browser and the Mi Browser, this behavior can be explained by the missing support for CTs. But Firefox, Tor Browser, and Samsung Internet opened CTs without notifying the user. Samsung Internet even opened full screen TWAs without notifying the user at all. Thus, if the displayed website looks similar to the previous Android app, the user has little chance to perceive the context switch. Tor Browser deletes its state after closure; therefore, the shared state only works if two instances run concurrently.

On all devices and browsers except Tor Browser, CTs and TWAs left traces. Browsers that spawned a new instance instead of opening the URL in a CT persisted the opened tab after the app was closed. All URLs opened in either CTs or TWAs left entries in the browser history. While the CT history entries contained the full URL, the TWA history entries only contained a title and the visited domain. Thus, they did not display that the app passed data to the website via URL parameters. In addition, we verified that our two test apps had access to the shared browser state on all devices and browsers. Browsers known for their privacy efforts, such as Firefox, Tor Browser, and Brave, are equally impacted as regular browsers.

> **Take-Away #1:** Shared Storage for apps via CTs and TWAs is possible across all configurations. Although only some browsers implement TWAs, most systems launch Chrome's TWA if the default browser lacks support.

4 Persistent Cross-App and Web Tracking

We propose HyTrack, a tracking technique that merges the full scope of web tracking into the native Android app context. Consequently, it allows tracking providers to track users effortlessly across apps and websites. This presents a severe new threat to user privacy on Android smartphones. Let us first present our threat model (Section 4.1) and the core concept of HyTrack (Section 4.2), followed up by an improved version of it to increase its impact (Section 4.3).

4.1 The Threat Model

HyTrack involves three parties: a tracking company, app developers, and the user. The tracking company wants to track users across different apps and websites. They provide a programming library and advertise it to developers of apps, e.g., for monetization purposes. The library initialization instigates the tracking.

Multiple distinct developers independently integrate the library from our tracking company into their apps. They follow the instructions to use the library and, upon initializing it, unwittingly start the tracking. We do not assume active collaboration between different developers or between developers and the tracking company beyond the inclusion and usage of a seemingly harmless library based on its documentation.

Finally, the user installs and uses multiple apps on their device, some of them include and use the tracking company's library. The user is unaware of which libraries are included in apps and, consequently, of the ongoing tracking. Therefore, they use the apps as they would any other.

Our threat model goes *beyond* the classic *app attacker model* [21] of unwanted apps and has less strict assumptions. The user intentionally installs the app on their device for their functionality and uses it accordingly. Thus, the app is a *wanted application* and not, as commonly assumed, a potentially unwanted one. Consequently, the user does not need to be tricked into installing or using malicious apps to launch the tracking. Therefore, deploying HyTrack at large is significantly easier than most threats to the mobile ecosystem.

4.2 Core Idea Behind HyTrack

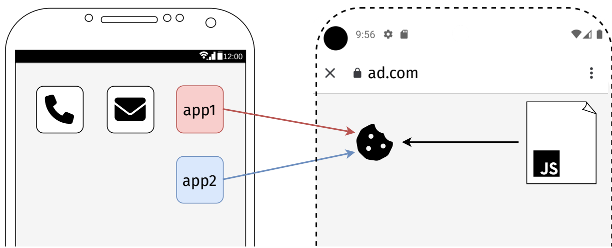

Figure 1: HyTrack: Different apps open Custom Tabs to the same website and use its cookies as persistent shared storage.

One fundamental aspect of Custom Tabs (CTs) and Trusted Web Activities (TWAs) is their sharing the full state with the browser's native storage, which we leverage for HyTrack. This implies that cookies persist across multiple sessions and even across apps. We leverage this design choice to set an identifier that the tracking library can reuse across different apps and websites to identify the user to our tracking company.

To set the tracking identifier, the library opens a Custom Tab with their tracking domain, here, ad.com. The requested URL encodes tracking data, such as the app's name and possibly other information useful for building a profile.

Upon receiving the data, if this was the device's first request, the server generates a unique tracking ID and associates it with the user. The server subsequently sets the ID as a cookie value in the response. If the user interacted with ad.com before, this cookie already exists, and the browser sends it with the request. The server can now link the newly received tracking data to the user's existing profile. As Custom Tabs share the browser's state across apps, all apps, including their included third-party library, therefore use the same tracking identifier. This also implies that this cookie is used during regular browsing on the smartphone. Consequently, by using the shared browser state to store the identifier, we get the link to tracking on the Web without any additional synchronization effort. Thus, not only is cross-app tracking now trivially possible, as is tracking the user across both native apps and the web using the same identifier.

> **Take-Away #2:** Leveraging the shared browser state, HyTrack enables cross-app tracking that even spans into the Web.

4.3 Disguising the Tracking Technique

A naive implementation of HyTrack is directly visible to the user. As soon as an app opens a Custom Tab, Android moves the app to the background or overlays it with the Custom Tab. In either case, the browser window displays the URL, i.e., ad.com, to the user. Additionally, an app cannot close a Custom Tab programmatically, and the user needs to close it manually to return to the app. Both of these aspects strongly indicate to the user that they have left the native app's context and are interacting with web content [22]. This could lead to suspicious users recognizing the tracking, or annoyed developers removing the library due to interference with the user experience. We found anecdotal evidence of this notification resulting in bad Google Play Store reviews in the Chrome bug tracker [22]. However, multiple options to obfuscate the ongoing tracking exist, hiding it from both the user and the app's developer.

Improving the User Experience After launching the tracking, the app is open in the background while a Custom Tab with the tracking domain, here ad.com, is displayed in the foreground. This is a noticeable interruption of the app's user experience. In the following, we present two options to make the user experience impact less noticeable with different strengths in terms of their tracking capabilities.

First, if the tracking shall be completely invisible to the user, Beer et al. [23] provide techniques to hide Custom Tabs and their content. To allow measuring user engagement with the displayed page, Custom Tabs offer to register callbacks that are invoked when navigation events occur. When the Custom Tab is shown, the `TAB_SHOWN` callback fires, and the library starts another activity. As activities on Android are layered on

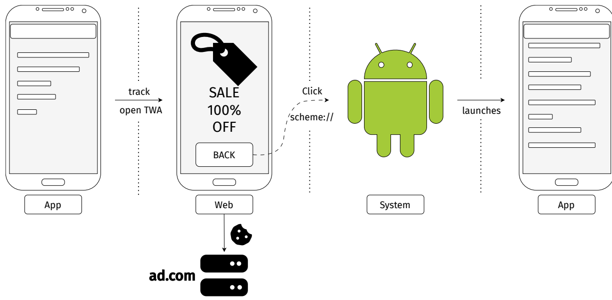

Figure 2: HyTrack can be seamlessly integrated into an existing app via Trusted Web Activities and a back-button.

top of each other, this new activity hides the previously opened custom tab. Combining this with a custom implementation of the back button functionality by the overlaying activity, the Custom Tab can be hidden completely from the user. This approach does not impact the user experience but also does not have a way to pass information back to the app, which is required to build more impactful variations of HyTrack later on.

The first and most simple way to disguise the tracking is to stylize the landing page of the tracking domain. It could display a loading or splash screen matching the app's user interface, as is commonly encountered upon launching mobile apps. This is easily achieved by selecting a matching stylesheet based on information such as the app used to invoke the Custom Tab, which can be passed via a URL parameter.

Furthermore, returning to the app requires the user to close the Custom Tab explicitly or to press the back button. We now discuss how it is possible to integrate the Custom Tab into the flow of the app and even establish a back channel. Web content can provide ways to navigate back to the native app, removing the need for the user to close the custom tab or navigate back manually via the OS's back gesture or button.

This requires the app to register a custom URL scheme (Section 2.2) and ad.com to display a button that opens a link using the said scheme. Common *Single-Sign On* (SSO) libraries such as "Login with Facebook" [24] use the same flow of user interaction and, thus, if executed well, it would not raise suspicion. Such custom scheme links also serve as a back-channel. As custom schemes allow encoding parameters, even when called from a Web context, ad.com can transmit the tracking identifier back into the native domain. Figure 2 provides an example for this app flow. Please note that for this to work reliably, the registered scheme has to be unique to the app. If another app registers the same scheme, the user has to choose which app to open.

Seamless Integration Using TWAs Our previous proposals already decrease the recognizability of the tracking and consequently improve the user experience. However, the URL bar remains visible as long as we are using Custom Tabs, providing a visual indicator that they interact with Web content instead of the native app. We can remedy this by using a Trusted Web Activity.

Trusted Web Activities always render in full screen and do not show any browser UI elements, such as the URL bar. Therefore, if the page is using the same design language as the app, transitions from native app to web content and back become nearly invisible to the user. The only remaining indicator is a little notice rendered by the browser that the currently displayed content is web content [22]. However, this notice varies greatly depending on the Android flavor and configured browser combination, as shown in Section 3. The Digital Asset Link (DAL) association required to launch a TWAs are indicators for privacy researchers that an app is connected to a tracking domain and vice versa, signaling the use of HyTrack. However, even this link can be obfuscated further. Instead of statically including the required JSON data that constitutes the DAL into the app, the tracking library can dynamically add any DALs needed at runtime. Using regular Java obfuscation techniques [25, 26], the corresponding code and generated origins can be obfuscated even further. To hide the website's association, it can modify its DAL JSON by adding the app dynamically based on certain conditions. To do so, the tracking code can conduct a pre-flight request to the web server, signaling an upcoming DAL request. The server can then return the desired DAL for the next time unit or the next DAL request from the sender's IP address. This allows the website to declare its trust relationships selectively, making it invisible to requests such as ours. However, these techniques are obfuscation techniques, which can be reverse-engineered with sufficiently advanced techniques.

> **Take-Away #3:** Leveraging Trusted Web Activities, the tracking can be hidden from the user while passing the tracking ID to the app. Both the website and app can establish the DAL dynamically.

4.4 Distributed Persistence of Tracking Cookie

Using the intent-based back channel, the tracking library also has access to the tracking identifier used on the Web. It can then persist the ID to either the app-specific storage or as a shared preference [27]. This allows the tracking ID to persist even if the cookie is removed from the shared browser profile or the browser is changed. When opening ad.com in a CT or TWA, the library can pass the identifier from the app storage to the tracker. This makes it increasingly challenging for users to remove this tracking identifier, as they would have to remove the cookie from both the browser profile and the app's internal storage. Once multiple apps include the tracking library, the situation gets even worse. As they all receive and store the tracking ID, the user would have to simultaneously clear each app-specific storage. If the library uses the TWA based disguise approach, whether an app includes the tracking code is not visible to any user and requires reverse engineer-

ing of the app to unveil. Even worse, Android, by default, backs up shared preferences to the user's Google Drive and restores them upon reinstalling the app [28]. Consequently, the tracking ID is now persistently linked to the user's Google account and is restored even upon the most drastic measures, such as resetting or changing the phone. Persisting a cookie in the local storage of one app and its backup is demonstrated in Figure 3 in steps ①-③. After the user clears their cookies (step ④), a TWA pushes the cookie back to the browser's cookie jar (step ⑤). Therefore, cookies created with HyTrack are even more persistent than the previous state of the art in cookie respawning, the evercookie [2, 29].

Take-Away #4: HyTrack allows resurrecting the tracking ID from any app using the tracking code, the web, or even automated Android backups.

Trackers using HyTrack can also share their data with other third parties over channels such as *Cookie Syncing* [29–31], increasing HyTrack's impact.

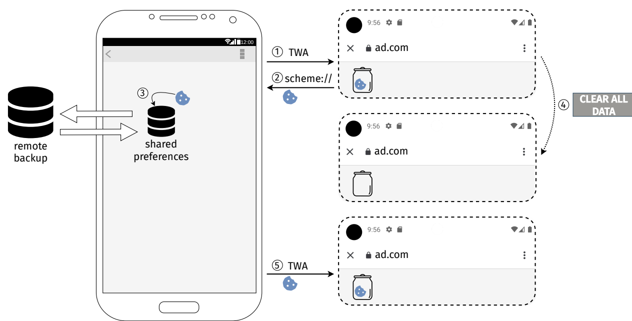

Figure 3: Cookies can be restored through the app storage.

4.5 Summary

Our novel tracking technique, HyTrack, has three essential features that outline its severity and privacy implications.

Firstly, HyTrack *combines cross-app and web tracking* by leveraging the smartphone's default browser state. It enables the tracking provider to not only track users across different apps that use the same library, but also reach into the user's regular web browsing. As the tracking identifier persists in the browser's cookie storage, the browser sends it both for all Custom Tab or Trusted Web Activity requests and during regular browsing on the smartphone. Thus, any tracking by the same company on the web can inadvertently leverage the same tracking cookies, bridging the gap between in-app tracking and web tracking. Even recent browser mitigations aimed to combat cookie-based tracking, such as storage partitioning and redirect tracking protection, released in Chrome in Q4 of 2024, will not protect against cross-app tracking as HyTrack uses first-party cookies. Secondly, HyTrack can be *seamlessly integrated into the app UI*. While naive usage of Custom Tabs and Trusted Web Activities is visible to the user, we presented multiple options to disguise the context switch between native and web content. As a result, the tracking will not be recognizable to the user, preventing them from taking precautions or preventative measures. Lastly, HyTrack allows to *resurrect deleted identifier* due to its access to not only the browser storage but also the app storage. Thus, it can synchronize identifiers that have been deleted from either storage unless the user conducts a thorough purge across the browser and all involved apps. Thus, the identifier persists even if the user deletes only the browser cookies or the apps involved in the tracking. This has severe implications for the user's ability to remove any stored tracking identifiers and regain control over their privacy.

5 Measurement Methodology

To study if there are any real-life deployments of HyTrack, we conduct one web and two Android app measurements. In the web measurement, we assess the usage of DALs on regular and known tracking websites. As our Android app measurements, we conduct a dynamic traffic collection of Android apps to gauge if we can already observe evidence for HyTrack and analyze how many apps include code to launch Custom Tabs as well as Trusted Web Activities.

5.1 Web Measurement

Based on a web measurement, we want to understand the prevalence and distribution of DALs across websites. If tracking with HyTrack is deployed via a TWA, tracking providers like ad.com have to list all connected apps in the hosted `assetlinks.json` (ref. Section 4.3). We analyze the web for DAL from three viewpoints: 1. The general Web 2. Tracking and advertisement entities from the Web 3. Tracking entities focussed on the mobile domain.

We collect the available `assetlinks.json` for the Tranco 500 000, based on the list G62VK, to measure the general frequency of DAL usage across all kinds of popular websites. As the Tranco list focuses on popular websites that do not necessarily launch sophisticated tracking attacks, we also analyzed two privacy-focused host lists. Our target list consists of the EasyPrivacy list [32] prepared for hostname-based blocking by `firebog.net` from July 1st of 2024. Compared to the regular easy privacy list used in popular ad blockers, such as AdBlock Pro or uBlock Origin, all rules specific to page content, such as CSS element identifier, rules matching specific JavaScript files, and similar, are removed, and we are left with a list consisting of hostnames only. The EasyPrivacy list is commonly employed for ad blocking on the web, and it is strongly biased towards web-based trackers.

To add the mobile perspective, which might have different entities [33], we also collected the `assetlinks.json`

for websites listed in the exodus tracker list [34], a tracking list focused on the mobile ecosystem. Here, we include all trackers with a known network signature, which we used to extract our domain list. Our collector follows best practices, i.e., identifying us as researchers with contact information to opt out of further visits via a custom header and only access the well-known path for the resource as required by Android.

5.2 Dynamic App Measurements

The goal of our two dynamic app measurement studies is to collect and subsequently analyze the outgoing and incoming traffic of popular apps. Any cross-app cookie sharing, a strong indicator for HyTrack, would appear in the collected traffic. Consequently, based on our measurement, we can assess if HyTrack is already used and by whom.

Quantitative Analysis Our large-scale dynamic measurement consists of a 60 second no-touch traffic collection. This means we install and start each app on a rooted Android phone. We did not disable the gaid. We then let the app run for 60 seconds without any further interaction while we intercept the traffic using mitmproxy [35]. To deactivate any SSL checks, we use Frida with Objection [36, 37]. Our measurement setup is automated using the framework presented by Koch et al. [38]. We are interested in both outgoing traffic and responses, so we extended their framework to collect incoming traffic.

Qualitative Analysis Since the no-touch analysis is fully automated and interaction-less, we additionally measure selected applications while manually interacting with them to uncover deeper functionality.

We selected the apps for manual apps from the set of apps that have CT capabilities and registered a unique scheme, informed by the static app measurement (Section 5.3). We selected the apps for manual testing by statically extracting URLs from the app dataset. Using only those on the Exodus or Easy Privacy list while excluding Google domains, we selected apps such that each encountered tracking domain in the set is contained in at least four apps, if possible. With this condition, we split this dataset into two buckets to run the experiment on two Google Pixel 6A with Android 14. The experiment was conducted by two researchers.

We manually interacted up to 15 minutes with the applications until they were exhaustively explored. Where possible, we logged in to the app. To reduce the impact of our testing on users and systems, we tried to avoid optional mechanisms that interact with other users, e.g., we did not submit comments. If apps crashed or did not work correctly, we added additional apps. The final set successfully covered 63 known tracking hosts as described above.

5.3 Static App Measurements

While the web measurement provides us with the web perspective on DAL deployment, our static analysis of Android apps does the same but from the app perspective. Additionally, we want to understand how prevalent CT and TWA usage is already across the Android ecosystem. Our analysis is based on androguard [39]. We forked the pipeline of Beer et al. [23] and adjusted it for our needs. In summary, we have three goals: a) measure CT and TWA usage, b) identify static DALs and dynamic DAL generation in apps, c) extract registered schemes for CT and TWA related activities.

CT and TWA Usage To distinguish between pure CT usage and CT usage via a TWA, we adopt the tooling of Beer et al. [23] to exclude apps using TWA features from counting towards CT usage. Their analysis looks for three factors indicating usage of CTs: a direct call to launch a Custom Tab, usage of Custom Tab-specific functions, and strings that fuzzily match the CT related class name. We expand the analysis with our improved search for TWA usage that is based on three factors: indicator 1): we check if an app directly contains a TWA launcher activity, indicator 2): we search for calls to library functionalities that launch TWA, including the deprecated but still used `TrustedWebUtils`, and, indicator 3): we search for code explicitly creating an intent to start a TWA.

To realize indicator 1), we leverage androguard's feature to extract all activities of an app and check if it includes the known TWA launcher activities. For indicator 2), we use androguard to extract all calls to functions that can start a TWA but exclude matches inside Android or Browser internals to only include usages that are in the app's main code base. Finally, for indicator 3), we search for usages of the TWA extra string, i.e., identify manual TWA launches. Again, we filter the results to exclude matches inside Android or Browser internals.

Static and Dynamic DALs DALs contained in an app have the same JSON structure as DALs on the web and are thus easily recognizable. Thus, we search the app's string resources file for the DAL pattern, as well as within the manifest and the app binary itself. For hits in the binary, we perform an additional manual inspection and exclude apps that are browsers, as browsers that implement the Trusted Web Activity protocol must naturally include DAL-related strings.

As it is possible to dynamically add a DAL at runtime, our static analysis also takes such code patterns into account. We use androguard to search for calls to the corresponding function *setAdditionalTrustedOrigins*. Additionally, we search for usages of the ADDITIONAL_TRUSTED_ORIGINS intent, which can be used to dynamically extend the trusted origin list when launching a TWA manually.

Registered Schemes Our final goal is to extract any scheme(s) registered by apps via an intent filter. To achieve this, we use androguard to extract all activities defined in an app. For every identified activity, we process the registered intent filters and only keep those that define the action VIEW and are of the BROWSABLE category. Both attributes are strict requirements to be called from the web. For every remaining filter, we extract the defined schemes.

6 Results

Let us present the results of our web measurement of our two app measurements. The app measurements are based on a dataset generated using the tooling provided by Koch et al. [38]. We downloaded the top 200 apps from the Google Play Store across the 32 main app categories on June 24th, 2024. This led us to a dataset of 4441 distinct apps.

6.1 Web Measurement

In total, we attempted to retrieve the `assetlinks.json` for 563 839 different hosts from the three lists. The two privacy lists contained many hosts at specific subdomains. Thus, we also crawled the eTLD + 1 hosts from these lists. Table 3 summarizes the results of our analysis of the crawled `assetlinks.json` instances.

Table 3: Dynamic Asset Links collected on the Web

	Easy-Privacy	Easy-Privacy*	Exodus Privacy	Exodus Privacy*	Tranco 500k	# Unique of All
# entries	37 685	25 969	744	398	500 000	563 839
Has DAL	1490	1660	8	11	12 798	14 744
DAL $\geq$ 2 dev.	11	58	2	4	531	545
DAL $\geq$ 3 dev.	0	14	0	1	89	92

*: Collected `assetlinks.json` from eTLD+1.

Of the unique crawled hosts, 14 744 defined valid DALs to Android apps. If apps or SDKs in the wild use Trusted Web Activities to track users across apps, we would expect to find a website that has digital asset links to multiple different apps to allow these apps or embedded SDKs to open the website in a TWA. We would expect these apps to come from different developers because there are easier communication channels between apps from the same developer. Therefore, we deduplicated the digital asset links of each website by the same app developer. We used the AndroZoo Google Play metadata [40] (acquired on June 18th, 2024) to look up the developer name, contact mail address, and URL for each package ID. We grouped two apps as belonging to the same developer if either developer name, mail address, or URL are non-empty and equal. The third row in Table 3 lists the number of digital asset links with apps from two or more different developers.

Next, we manually inspected 92 websites that link apps from three or more different developers. We search for indicators of HyTrack by searching for apps that link to the same website for tracking purposes. We are not interested in cases where they are linked due to shared code bases, e.g., same developers or software white labeling. Thus, we visited the website and the app's download pages in the Play Store to determine if the website hosts content belonging to the applications. All observed clusters were benign cases of apps sharing the same backend. For example, we observed financial software developers providing the backend used by several banks' banking apps.

6.2 Dynamic App Measurement

The mission statement of our dynamic measurement study is to find indicators that HyTrack is already being used. To achieve this, we conducted both a large scale quantitative analysis and a smaller scale qualitative dynamic measurement. The quantitative measurement did not involve any interaction with a started app, whereas the qualitative analysis involved the manual exploration of the scrutinized apps. Any such usage would be visible in our collected traffic as two independent apps sharing sufficiently complex cookie values. We extract all sent and received cookies from each app to identify shared cookie values and calculate the intersections of values across apps.

Quantitative Measurement We ran our measurement over the course of two weeks and fully collected traffic for 4037 apps, using a single Pixel 6A running Android 14, rooted using Magisk [41]. We observed a total of 152 907 requests with a mean of 35.09 ($\sigma = 69.54$) per app. Out of these, we successfully intercepted 114 801 requests to 7461 domains and failed to intercept 38 106 requests to 599 domains. This means that we, by average, were able to intercept 26.34 ($\sigma = 55.69$) requests per app and failed for 8.74 ($\sigma = 41.07$). Interception failures and unsuccessfully analyzed apps can be explained by certificate-pinning not covered by objection and factors such as the instrumentation process via Frida. The most frequently successfully intercepted domain was graph.facebook.com (8395 requests) and the most frequently failed domain was digitalassetlinks.googleapis.com (7373 requests). Table 4 summarizes the statistics of our successfully intercepted requests, split across traffic directed to tracking and non-tracking endpoints. The tracking classification is based on matching domains against the Exodus tracker list [34]. This already excludes background traffic of the phone itself.

Overall, we observed 45 710 cookies being sent and 10 492 received. Most cookies are being sent to and received from non-tracking domains. We observed 1518 different cookie names with 344 being used across different apps. The most common cookie name is __cf_bm, which is Cloudflare related. According to Cloudflare's website, "The __cf_bm cookie is necessary for [their] bot solutions to function properly."[1]

When looking at the cookie values, we observed 6944 distinct cookie values, with the most popular being "". Overall 96 values are shared across apps. We discarded all trivial and obvious non-identifier values to extract potential identifiers, resulting in 29 values remaining shared across apps. We give a complete list of all discarded cookie values in Listing 1 the Appendix C.

[1] https://developers.cloudflare.com/fundamentals/reference/policies-compliances/cloudflare-cookies/

Table 4: Overview of the collected traffic of 4037 apps. The categorization of non-tracking and tracking is based on Exodus.

	Outgoing						Incoming					
	Requests			Cookies			Response			Cookies		
	#	∅	σ	#	∅	σ	#	∅	σ	#	∅	σ
All	114 801	26.34	55.69	45 710	10.49	152.24	115 409	26.48	57.85	10 492	2.41	16.67
Non-Tracking	75 324	17.28	49.85	45 159	10.36	151.72	76 619	17.58	52.36	8629	1.98	15.76
Tracking	39 477	9.06	18.07	551	0.13	2.77	38 790	8.90	17.80	1863	0.43	2.62

As the cookies need to belong to the same domain to establish a first-party context, we discard cookies with shared values from different domains. This reduces the set of shared cookie values to 18. An additional requirement to fit our described tracking methodology is that all apps must at least send the value and use the user agent of the default browser, further reducing the number of cookies down to 9. We inspected all related applications and filtered for those that launch a Trusted Web Activity or Custom Tab at startup. 7 applications fit these criteria, out of which none launched a Custom Tab. Further inspection revealed that only 3 applications provide an activity besides the Trusted Web Activity-launcher with the remaining 4 applications being pure web-wrappers. These three applications containing further activities are also mainly wrapping a website in a Trusted Web Activity. The additional activities are merely used to handle, e.g., payment processing. This means that the exposed traffic behavior is not an indicator of HyTrack but merely a reflection of their website's behavior.

Qualitative Measurement For the manual analysis, we interacted with 95 applications, out of which we could successfully explore 65. The remaining applications crashed on startup or during interaction, required an update, or did not fully load. We collected a total of 81 920 requests and 78 457 containing 97 711 and 23 308 cookies. We observed requests to known tracking domains in 89 apps with a total of 18 849 and 19 514 responses.

We applied the same analysis process as during our large-scale analysis. This revealed a set of 26 apps that shared a cookie with at least one other app in the set. Using these shared cookies as starting points, we further inspected the corresponding apps. We did not identify instances of HyTrack in these apps. The observed behaviors were due to traffic in regular browsers or CTs launched by clicks, loading pages, and conducting regular web tracking without perceived linkage to the app.

6.3 Static App Measurement

With our static analysis, we successfully analyzed 4376 apps from the dataset of 4441 apps. The measurement was conducted on a machine with an AMD EPYC 7702P 64-Core CPU with 512 GB of RAM, taking about two hours. Figure 5 in the appendix provides a visual overview of our results.

CT and TWA Usage We found 1537 apps with a call to launch a Custom Tab, 1546 apps that use Custom Tab-specific functions, and 1868 apps containing a string that matched the fuzzy class name. Concerning TWA usage, we found two applications that define TWA activities in their manifest, seven apps with code to launch a Trusted Web Activity via a helper library call, and 184 apps that used the TWA intent extra. In total, 190 apps seem to use TWAs.

Static and Dynamic DALs We identified 56 apps that included a DAL. All matches contained only a single DAL JSON, with some containing multiple links. In total, we found 244 links, with a mean of 4.52 (σ 16.42) across all apps. We discovered all DALs inside the string resources. Our search within the binary code resulted in one non-browser application that contained our looked-for indicator: the Google Analytics App[2]. A manual inspection indicates that the app can check if a website has configured a DAL. *No pair* of apps contained DALs linking them to the same website.

Our search for the dynamic addition of DALs during runtime revealed 190 different applications that set an additional trusted origin. 33 apps call the function, while 5 apps extend the list via their manifest and 152 contain the respective intent extra flag.

Registered Schemes We identified 2896 apps with web-launchable activities and extracted the schemes of their intent filters. 3834 schemes were used by only one application, 88 schemes were used by 2 apps, 39 by 3, 19 by 4, and 61 by more. We are most interested in the unique schemes, which can be used to reliably link back from the CT or TWA. Note that this is a lower bound for this specific property since applications could combine more popular schemes with the 'intent:' link syntax, which requires a specific app ID.

Overall, schemes that are only registered by at most one application are the most common case, occurring for 3834 schemes. In total, 1927 apps have a filter for a unique scheme. This is useful for HyTrack, as unique schemes can be used to launch one specific app in the implementation of the TWA's back button (ref. Section 4.3). Out of those apps, 1718 apps registered HTTPS in an intent filter, while 809 listened to HTTP. Recall that HTTP schemes must use a DAL to launch apps in recent Android versions.

A visual inspection revealed clusters of unique schemes that shared a prefix. The most noteworthy cluster belongs to

[2] `com.google.android.apps.giant`

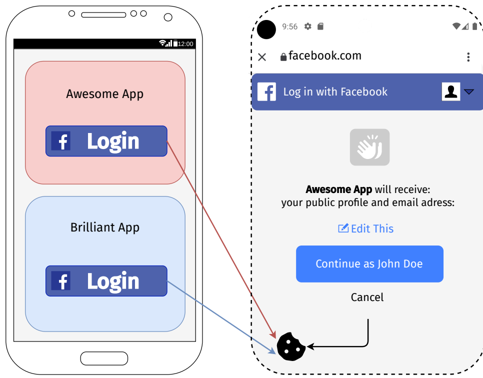

Figure 4: Facebook SDK uses Custom Tabs for login.

the Facebook SSO, with 480 different but similar schemes. Each scheme is unique but consists of the string 'fb', followed by a numerical ID. These schemes are used by apps that implement a 'Login with Facebook' functionality. The Facebook SDK uses Custom Tabs with this custom scheme to allow a seamless return to the app from the login page CT [24]. To use the Facebook SSO, developers register an activity with the SDK linked to by the custom scheme. The corresponding SDK function opens a Custom Tab to the login page when the app calls the Facebook SSO. After logging in, the user is redirected back to the app activity using the unique scheme. This activity then passed the data deeper into the SDK for further processing. We visualized the workflow of the Facebook SSO in Figure 4. It leverages the shared browser state of Custom Tabs to share access to the Facebook account across the browser and multiple apps.

The second most prevalent cluster belongs to the Google Tag Manager and starts with 'tagmanager.c'. We found 171 apps that register a unique scheme with this prefix. These schemes and the corresponding activities are used to implement debug and preview functionality for the Google Tag Manager [42].

Combining the static analysis results, we found 840 apps that possess both Custom Tab capabilities and a unique scheme. Therefore, we found that over 20 % of the analyzed apps already employ all the requirements for a tracker to deploy HyTrack right now.

7 Discussion

HyTrack has the potential to make tracking inescapable on Android. Our description of the tracking technique (Section 4) and demo published alongside this paper (Section 10) show that the execution is straightforward. Only basic Android functionality is utilized by HyTrack. We also show how to integrate the tracking into the basic app flow without undue fuss. As it is already common nowadays to switch between an app and a Custom Tab or Trusted Web Activity for actions such as single sign-on (SSO), leveraging such a UI for tracking would not raise suspicion among the common Android user. This is notwithstanding the ability to simply hide the Custom Tab as suggested by Beer et al. [23] and discussed in Section 4.3. Exposing users to HyTrack also only requires minimal collaboration between the app developer and the tracking company. Our measurements show that app developers are already used to SDKs requiring the addition of a custom scheme, such as for the Facebook SDK [24]. Additionally, studies from the web have indicated that developers are eager to implement new requirements to enable tracking techniques, e.g., for CNAME cloaking [43]. Furthermore, past work has shown that developers consider monetization SDKs essential [44], likely limiting their questioning of the integration documentation.Consequently, the ingredients for a wide adaption of HyTrack are already present, and it can be rolled out by pushing a new version of an SDK without further action required from the app developers.

The impact on users in case of widespread deployment would be severe. Despite the user deactivating the gaid, cross-app tracking would become trivially possible again, with all its privacy implications. Moreover, we assume that trackers prefer HyTrack over the gaid as it is more powerful. The ability of the user to defend against HyTrack is significantly impacted by the ability of the tracker to link cookie and app storage as detailed in Section 4.3 and 4.5. Even if the user deletes the browser's cookie storage or even their whole browser profile, they only need to start a previously run app that includes the library launching HyTrack for the tracking provider to restore the deleted tracking identifier. Even switching the default browser does not get rid of the stored tracking identifier, as the persistence mechanism is not based on a browser bug but relies on basic functionality on Android, i.e., CT and TWA support. Thus, the only complete defense is deleting all apps deploying HyTrack as well as clearing all previously used browser caches. As it is not easily visible which apps include certain libraries, this is an insurmountable task for the common user. Even then, as the tracking identifier can persist in the app storage segment that is automatically backed up if Google cloud backup is enabled, there is no remediation, nor is it changing to a new device.

Luckily, we have not found any deployment of HyTrack in the wild. However, the building blocks are already in place. While the general usage of SDKs is omnipresent [44], a third of the apps in our dataset contain code to interact with Custom Tabs and 40% register an app-specific scheme, required to channel back information into the native code. Consequently, a third of our dataset has all the prerequisites to start using HyTrack available.

The stated goal of Custom Tabs and by extension Trusted Web Activities is to "increase user experience" by "automatically sharing the state and features offered by it [the user's browser]" as well as the seamless transition between native and Web content [13]. Based on these declared utilities, we de-

rive three goals that any mitigation must keep intact: goal 1): Support all features of the Web platform goal 2): Do not break the seamless integration, and goal 3): Make shared state available to the Custom Tab or Trusted Web Activity. As HyTrack is based on legitimate usage of the Custom Tab and Trusted Web Activity APIs, simple remediation is not possible and would come with significant utility impact for both Custom Tabs and Trusted Web Activities as it would need to break at least one of the listed goals.

As of now, we see two possible avenues for mitigation: *Browser State Partitioning* or *Forced User Interaction*. Each solution impacts utility differently. Browser state partitioning would require each app to use a separate browser profile. This would ensure that there is no cross-app tracking capability, as every app has its values for the tracking cookie. It would also keep the seamless integration intact, as no changes to the UI are required. However, it severely degrades the utility by also preventing non-tracking uses of the shared state. This would break benign uses such as *Login with Facebook*, which would be a significant breaking change. This approach would keep goal 1) & goal 2) but directly violate goal 3).

The forced user interaction mitigation requires explicit user permission to launch a CT or TWA. Here, the URL to be launched and a warning message of the potential implications are displayed to query the user's consent before proceeding. This makes tracking attempts explicit to an expert user but breaks the seamless integration, i.e., goal 2), while being open to both user fatigue and obfuscation. Repeated permission prompts were shown to be ineffective in querying informed consent decisions [45, 46]. Similarly, determining whether a displayed URL is safe to visit or might expose the user to undesirable behavior is an impossible task for all but the most motivated and aware users. Using techniques from phishing to disguise domains as benign [47, 48] or even use non-descriptive domain names leaves the user in the dark about the danger of being tracked. Additionally, we found reports of developers complaining about the current TWA notices [49], which are less intrusive than any possible consent dialog. Let us explore a few other mitigation approaches:

Limiting CT& TWAs to one domain: By restricting the possible domains, an app can open in CTs or TWAs to one domain tied to the developer; we can prevent the data flow from the app to arbitrary tracker domains. Other domains are not accessible in this mode. This would allow developers to use CTs and TWAs to augment their apps with their own web content. For example, developers could develop crucial components like account management only in the web application. This reduces complexity for them because they do not have to re-implement every feature for mobile, and due to the CT technologies, users can easily access the webpage from the mobile app. However, this drastically reduces the usefulness of Custom Tabs and Trusted Web Activities in other use cases. Features like *Login with* would not be possible anymore. It is a variation of browser state partitioning since it only makes one site's state accessible.

Permission System or Browser Option: Android could introduce new runtime permissions required for apps to use CTs or TWAs. Alternatively, browser vendors could allow users to disable the feature altogether or for specific pages. This moves the burden on the users and is a variation of forced user interaction. It, therefore, suffers from the same banner fatigue and obfuscation issues.

Blocking Traffic: Until the issue is fixed systematically, we recommend that users use a browser that supports ad blocking and subscribe to privacy blocking lists to combat potential HyTrack deployments.

> 💡 **Take-Away #5:** HyTrack uses Browser and Custom Tab functionality as it is intended. Remediation requires fundamental changes to the Custom Tab idea.

Neither of the discussed solutions is workable as remediation, as both violate one of the three derived goals of Custom Tabs and Trusted Web Activities. As we could not detect any evidence that HyTrack is currently being deployed, we believe there is still time to develop and roll out mitigations. We are corresponding with Android, Chrome, and Firefox developers about potential mitigations via our submitted bug reports. We provide proof of concept apps and open-sourced our measurement tooling[3]. This enables future researchers to keep measuring the prevalence and testing novel mitigation approaches.

Limitations To achieve the seamless nature of HyTrack with TWAs, web content needs to be integrated into the app. Since the web content is hosted by ad.com, they would either have to provide custom designs for the supported apps, provide designs for all major UI frameworks, as we did, or even simpler serve some content, which is okay to break the app's design. For example, full-screen advertisements, loading screens, etc., often do not adhere to the regular app design.

We only found the necessary indicators that HyTrack can be deployed, but no direct evidence of any app doing so. However, our three measurements have certain limitations that contextualize the results. The web measurement can only retrieve trust relationships the website makes permanently available. If, for example, the obfuscation approach detailed in Section 4.3 is employed, our measurement misses these DALs and, consequently, the trust link between the website and its linked apps. Similarly, the static analysis presented in Section 5.3 is based on analyzing identifiers. Consequently, it can misclassify usages if the code is obfuscated with scrambled class names. In contrast, our dynamic measurement presents

[3]See the Open Science Considerations (Section 10).

more of a lower bound as we do not interact with all apps. It is possible that apps already employ HyTrack, but it is hidden deep in the app flow and is consequently not discoverable by our no-touch analysis.

8 Related Work

Tracking is an active research topic, and approaches can be grouped into stateless tracking using fingerprinting or stateful tracking by storing a persistent identifier on the user's device for identification.

Fingerprinting is an extensively explored research topic, with Eckersley [3] being the first to show that browser attributes such as User Agents can help to identify and track users. Their work was followed by improvements ranging from leveraging different browser attributes [8, 50–56], access to limited resources [57], browser extensions [58, 59], or even the underlying hardware [60, 61] or how well they transfer to the mobile domain [62]. Similar work has been done to conduct fingerprinting of mobile devices, leveraging API access [63, 64], hardware [5, 65–69], and software behavior [70]. Finally, Zimmeck et al. [71] have shown the feasibility and existence of cross-device tracking. Each of the cited works improved our understanding of how stateless tracking is possible by abusing the variety of devices and configurations. This contrasts with our proposed stateful approach, which leverages a fundamental feature of the mobile browser implementation. Our tracking approach works regardless of configuration changes or hardware access.

Regarding stateful tracking, Roesner et al. [72] have categorized such approaches. Based on these categories, Lerner et al. [73] conducted a longitudinal study to assess their prevalence over time, showing a sharp rise in overall tracking activities. As browser vendors try to mitigate the prevalence of third-party stateful tracking by restricting traditional tracking vectors such as third-party cookies [74–76], advertisement companies have reacted by shifting to either stateless tracking or leveraging first-party cookies [77]. Techniques such as CNAME cloaking [43, 78] or cookie ghostwriting [79, 80] and cookie syncing, also called cookie matching, [29–31] all allow to leverage first-party cookies to track users across the Web. Ali et al. [81] have devised a methodology to automatically assess whether newly added browser functionality can be used for stateful tracking. We add to this work by presenting a novel approach to abusing browser features to achieve persistent stateful tracking.

Mobile privacy was thoroughly discussed in previous work [38, 82–86]. Beer et al. [23] discussed the security implications of Custom Tabs and techniques to hide them. Although they discuss how CTs can be used to track user engagement, they do not discuss TWAs. With HyTrack, we propose a powerful tracking technique, disguised by TWAs.

9 Conclusion

Custom Tabs and especially Trusted Web Activities allow Android apps to freely intermix native and web content and even leverage the shared browser state for, e.g., Single-Sign on. This offers great potential for an improved user experience. For example, to connect an app with a Facebook account, the *Login with Facebook* library allows the use of credentials stored in the browser to authenticate, removing the requirement to enter credentials. However, blurring the lines between native and web content opens the door for web-based threats, such as tracking identifiers linking native and web profiles.

In this work, we present HyTrack, which is a novel tracking approach. HyTrack uses the shared state of the browser to track the user both in-app and across the web. Furthermore, it deploys techniques to disguise context switches between native and web content to hide its intention. By leveraging back channels and the OS backup functionality, we show that HyTrack is persistent across browser changes, attempts of cookie deletion, or even device changes. Thus, it is even more powerful and difficult to remove than the previous state of the art in cookie respawning, the evercookie [2].

We have confirmed that HyTrack can be executed on various Android flavors and browsers. While we have not found any indicators that HyTrack is used for tracking in the wild right now, all its building blocks are frequently used across the most popular Android apps. Therefore, the additional effort required to roll out tracking with HyTrack is minimal. Due to HyTrack reusing functionality used in many apps, such as registering app-specific schemes, it is not possible to mitigate without impacting the core functionality of Custom Tabs and Trusted Web Activities. This means rethinking the integration between web and native content is necessary to avoid exposing users to tracking identifiers that are almost impossible to remove.

Acknowledgments

We thank our shepherd and reviewers for their valuable suggestions and feedback. Furthermore, we thank Tobias Jost for his technical support and Jan Heuer for helping with the figures. We gratefully acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972 as well as from the European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206.

10 Open Science Considerations

We are committed to open science. We will open-source our toolchain for the web, dynamic app, and static app studies. We also provide a screen recording of a demonstration of

these PoCs in action, alongside a timestamp-based description, demonstrating the feasibility of HyTrack. We also provide a screen recording of a demonstration of these PoCs in action, alongside a timestamp-based description.

The code, demo, and description can be found at `https://doi.org/10.5281/zenodo.14718794`.

11 Ethical Considerations

We did not deploy HyTrack into any app store. We have not tested it on third parties. After concluding this work, we will delete all raw traffic collections to remove any user-generated content requested by the apps.

We disclosed HyTrack to Google via the Android Vulnerability Reward Program. Furthermore, we are in contact with the affected upstream browser vendors, disclosing our findings to the engine developers of the browsers tested browsers with Custom Tabs or Trusted Web Activities support. As of 2025-01 the Chromium team considers this issue a *won't-fix*, while the Firefox team discussed potential mitigations. Additionally, we contacted the vendors of downstream privacy browsers.

However, as detailed earlier, resolving this privacy threat is far from trivial and likely requires fundamental changes to the core Custom Tab functionality.

Therefore, weighing the harms of publishing this work against the benefits is detrimental. As harm, we identified the risk to user privacy that tracking providers and malicious actors could use our findings about HyTrack to deploy it themselves.

However, by publishing this work, we contribute to user knowledge. HyTrack is a risk users must be aware of to make informed choices about their browsing and mobile usage behavior. Only by knowing about it, users can act accordingly, e.g., by disabling features or switching to a browser with ad-blocking capabilities to use privacy-focused blocking lists. Additionally, publishing this work puts pressure on the browser vendors to reconsider their implementations and provide safer defaults. We only describe this issue while browser vendors ship features that can compromise users' privacy.

Additionally, we assume that other researchers and tracking companies will encounter this behavior. Even though we could not identify HyTrack instances in the wild, we can not be sure that tracking vendors and malicious actors are unaware of it or using it on people or apps outside our dataset. I.e., HyTrack might already be in use, putting users at risk. Therefore, we would harm user privacy by *not* publishing this work, as they could or might be already impacted by HyTrack in the future without their knowledge.

We consider the harm from publishing this work sufficiently reduced by the benefits of making users aware of the risk and allowing them to act accordingly, putting pressure on vendors to fix the issue systematically, and informing users that might already be impacted by HyTrack without their knowledge.

We urge all researchers working on similar issues to weigh the harms and benefits of their work as well before publishing it. Similarly, we want to encourage vendors to consider privacy implications early in the design of new (browser) features; privacy must be by design and considered early.

References

[1] "Advertising ID - Play Console Help," visited 2024-07-09. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/6048248?hl=en

[2] S. Kamkar, "Samy kamkar - evercookie - virtually irrevocable persistent cookies," visited 2024-07-05. [Online]. Available: https://samy.pl/evercookie/

[3] P. Eckersley, "How Unique Is Your Web Browser?" in *Privacy Enhancing Technologies Symposium*, 2010.

[4] A. Das, N. Borisov, and M. C. Caesar, "Do you hear what i hear?: Fingerprinting smart devices through embedded acoustic components," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[5] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[6] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Network and Distributed System Security Symposium*, 2019.

[7] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[8] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Krügel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *IEEE Symposium on Security and Privacy*, 2013.

[9] K. Mowery and H. Shacham, "Pixel perfect : Fingerprinting canvas in html 5," in *W2P*, 2012.

[10] "Browser fingerprinting: An introduction and the challenges ahead," 2019, visited 2025-01-10. [Online]. Available: https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead/

[11] "Common intents | android developers," visited 2024-07-02. [Online]. Available: https://developer.android.com/guide/components/intents-common#ViewUrl

[12] "Webview | android developers," visited 2024-07-02. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView

[13] "Overview of android custom tabs | web on android | chrome for developers," visited 2024-07-02. [Online]. Available: https://developer.chrome.com/docs/android/custom-tabs

[14] "Multi-origin trusted web activities | web on android | chrome for developers," visited 2024-07-02. [Online]. Available: https://developer.chrome.com/docs/android/trusted-web-activity/multi-origin

[15] Visited 2024-07-04. [Online]. Available: https://github.com/google/digitalassetlinks/tree/master

[16] "Google digital asset links | google for developers," visited on 2024-07-04. [Online]. Available: https://developers.google.com/digital-asset-links

[17] "Deprecate or remove twa intent processor | issue 12024 | mozilla-mobile/android-components," visited 2024-07-08. [Online]. Available: https://github.com/mozilla-mobile/android-components/issues/12024

[18] "Mobile browser market share worldwide," 2024, visited 2024-12-13. [Online]. Available: https://gs.statcounter.com/browser-market-share/mobile/worldwide

[19] "Antitrust: Commission fines google €4.34 billion for illegal practices regarding android devices," visited 2024-07-10. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/IP_18_4581

[20] Visited 2024-07-10. [Online]. Available: https://developer.android.com/reference/com/google/android/material/snackbar/Snackbar

[21] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on webview in the android system," in *Annual Computer Security Applications Conference*, 2011.

[22] "Implement 'running in chrome' toast for twa [40563338]," visited 2024-07-04. [Online]. Available: https://issues.chromium.org/issues/40563338

[23] P. Beer, M. Squarcina, L. Veronese, and M. Lindorfer, "Tabbed Out: Subverting the Android Custom Tab Security Model," in *IEEE Symposium on Security and Privacy*, 2024.

[24] "Android - facebook login," visited 2024-07-08. [Online]. Available: https://developers.facebook.com/docs/facebook-login/android/

[25] S. Rasthofer, S. Arzt, M. Miltenberger, and E. Bodden, "Harvesting runtime values in android applications that feature anti-analysis techniques." in *Network and Distributed System Security Symposium*, 2016.

[26] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications." in *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[27] "Data and file storage overview," visited 2024-07-04. [Online]. Available: https://developer.android.com/training/data-storage

[28] "Back up user data with auto backup," 2024, visited 2025-01-10. [Online]. Available: https://developer.android.com/identity/data/autobackup

[29] G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz, "The web never forgets: Persistent tracking mechanisms in the wild." in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[30] P. Papadopoulos, N. Kourtellis, and E. P. Markatos, "Cookie synchronization: Everything you always wanted to know but were afraid to ask." in *International World Wide Web Conference*, 2019.

[31] L. Olejnik, M.-D. Tran, and C. Castelluccia, "Selling off user privacy at auction." in *Network and Distributed System Security Symposium*, 2014.

[32] "Easylist - overview," visited 2024-07-04. [Online]. Available: https://easylist.to/

[33] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem." in *Network and Distributed System Security Symposium*, 2018.

[34] "Etip exodus," visited 2024-07-08. [Online]. Available: https://etip.exodus-privacy.eu.org/trackers/all

[35] "mitmproxy - an interactive HTTPS proxy," visisted 2024-07-04. [Online]. Available: https://mitmproxy.org/

[36] "Android | frida a world class dynamic instrumentation toolkit," visisted 2024-07-04. [Online]. Available: https://frida.re/docs/android/

[37] Visisted 2024-07-04. [Online]. Available: https://github.com/sensepost/objection

[38] S. Koch, B. Altpeter, and M. Johns, "The OK Is Not Enough: A Large Scale Study of Consent Dialogs in Smartphone Applications," in *USENIX Security Symposium*, 2023.

[39] Visited 2024-07-04. [Online]. Available: https://github.com/androguard/androguard

[40] M. Alecci, P. J. Ruiz Jiménez, K. Allix, T. F. Bissyandé, and J. Klein, "Androzoo: A retrospective with a glimpse into the future," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, 2024.

[41] "Download magisk beta," visited 2024-07-04. [Online]. Available: https://magiskmanager.com/magisk-beta

[42] "Google tag manager for android | google tag manager for mobile (android) | google for developers," visited 2024-07-10. [Online]. Available: https://developers.google.com/tag-platform/tag-manager/android/v5

[43] A. Aliyeva and M. Egele, "Oversharing Is Not Caring: How CNAME Cloaking Can Expose Your Session Cookies," in *ACM ASIA Conference on Computer and Communications Security*, 2021.

[44] A. H. Mhaidli, Y. Zou, and F. Schaub, ""We Can't Live Without Them!" App Developers' Adoption of Ad Networks and Their Considerations of Consumer Risks." in *Symposium on Usable Privacy and Security*, 2019.

[45] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," in *Symposium on Usable Privacy and Security*, 2012.

[46] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness." in *USENIX Security Symposium*, 2013.

[47] H. Hu, S. T. K. Jan, Y. Wang, and G. Wang, "Assessing browser-level defense against idn-based phishing." in *USENIX Security Symposium*, 2021.

[48] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Félegyházi, and C. Kanich, "The long "taile" of typosquatting domain names." in *USENIX Security Symposium*, 2014.

[49] "Make "opened in chrome" toast in twas configurable," 2019, visited 2024-07-10. [Online]. Available: https://issues.chromium.org/issues/41469302

[50] S. Boussaha, L. Hock, M. Bermejo, R. C. Rumin, A. C. Rumin, D. Klein, M. Johns, L. Compagna, D. Antonioli, and T. Barber, "FP-tracer: Fine-grained Browser Fingerprinting Detection via Taint-tracking and Multi-level Entropy-based Thresholds," in *Privacy Enhancing Technologies Symposium*, 2024.

[51] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "Fp-scanner: The privacy implications of browser fingerprint inconsistencies," in *USENIX Security Symposium*, 2018.

[52] A. Klein and B. Pinkas, "DNS Cache-Based User Tracking." in *Network and Distributed System Security Symposium*, 2019.

[53] K. Solomos, J. Kristoff, C. Kanich, and J. Polakis, "Tales of Favicons and Caches: Persistent Tracking in Modern Browsers." in *Network and Distributed System Security Symposium*, 2021.

[54] V. Mishra, P. Laperdrix, W. Rudametkin, , and R. Rouvoy, "Déjà vu: Abusing browser cache headers to identify and track online users," in *Privacy Enhancing Technologies Symposium*, 2021.

[55] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *IEEE Symposium on Security and Privacy*, 2016.

[56] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," in *Nordic Conference on Secure IT Systems*, 2011.

[57] P. Snyder, S. Karami, A. Edelstein, B. Livshits, and H. Haddadi, "Pool-party: Exploiting browser resource pools for web tracking," in *USENIX Security Symposium*, 2023.

[58] P. Laperdrix, O. Starov, Q. Chen, A. Kapravelos, and N. Nikiforakis, "Fingerprinting in style: Detecting browser extensions via injected style sheets," in *USENIX Security Symposium*, 2021.

[59] O. Starov, P. Laperdrix, A. Kapravelos, and N. Nikiforakis, "Unnecessarily identifiable: Quantifying the fingerprintability of browser extensions due to bloat," in *International World Wide Web Conference*, 2019.

[60] T. Laor, N. Mehanna, V. Dyadyuk, A. Durey, P. Laperdrix, C. Maurice, Y. Oren, R. Rouvoy, W. Rudametkin, and Y. Yarom, "DRAWN APART: A Device Identification Technique based on Remote GPU Fingerprinting," in *Network and Distributed System Security Symposium*, 2022.

[61] Y. Cao, S. Li, and E. Wijmans, "(cross-)browser fingerprinting via os and hardware level features," in *Network and Distributed System Security Symposium*, 2017.

[62] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the robustness of mobile device fingerprinting: Can mobile users escape modern web-tracking mechanisms?" in *Annual Computer Security Applications Conference*, 2015.

[63] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling, "Fingerprinting mobile devices using personalized configurations," in *Privacy Enhancing Technologies Symposium*, 2016.

[64] A. Das, G. Acar, N. Borisov, and A. Pradeep, "The web's sixth sense: A study of scripts accessing smartphone sensors." in *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[65] D. Cozzolino and L. Verdoliva, "Noiseprint: A cnn-based camera model fingerprint," in *IEEE Transactions on Information Forensics and Security*, 2018.

[66] A. Das, N. Borisov, and E. Chou, "Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures," in *Privacy Enhancing Technologies Symposium*, 2018.

[67] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable," in *Network and Distributed System Security Symposium*, 2014.

[68] J. Zhang, A. R. Beresford, and I. Sheret, "Sensorid: Sensor calibration fingerprinting for smartphones," in *IEEE Symposium on Security and Privacy*, 2019.

[69] A. Das, N. Borisov, and M. Caesar, "Tracking mobile web users through motion sensors: Attacks and defenses." in *Network and Distributed System Security Symposium*, 2016.

[70] L. Foppe, J. Martin, T. Mayberry, E. C. R. Rye, and L. Brown, "Exploiting TLS Client Authentication for Widespread User Tracking," in *Privacy Enhancing Technologies Symposium*, 2018.

[71] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara, "A privacy analysis of cross-device tracking," in *USENIX Security Symposium*, 2017.

[72] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web." in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[73] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016." in *USENIX Security Symposium*, 2016.

[74] "Firefox 86 introduces total cookie protection - mozilla security blog," visited 2024-07-08. [Online]. Available: https://blog.mozilla.org/security/2021/02/23/total-cookie-protection/

[75] "Prepare for phasing out third-party cookies | privacy sandbox | google for developers," visited 2024-07-08. [Online]. Available: https://developers.google.com/privacy-sandbox/3pcd

[76] G. Franken, T. van Goethem, and W. Joosen, "Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies." in *USENIX Security Symposium*, 2018.

[77] S. Munir, S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso, "Cookiegraph: Understanding and detecting first-party tracking cookies," in *ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[78] Y. Dimova, G. Acar, L. Olejnik, W. Joosen, and T. van Goethem, "The CNAME of the Game: Large-scale Analysis of DNS-based Tracking Evasion," in *Privacy Enhancing Technologies Symposium*, 2021.

[79] I. Sanchez-Rola, M. Dell'Amico, D. Balzarotti, P.-A. Vervier, and L. Bilge, "Journey to the Center of the Cookie Ecosystem: Unraveling Actors' Roles and Relationships," in *IEEE Symposium on Security and Privacy*, 2021.

[80] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos, "Cookie swap party: Abusing first-party cookies for web tracking." in *International World Wide Web Conference*, 2021.

[81] M. M. Ali, B. Chitale, M. Ghasemisharif, C. Kanich, N. Nikiforakis, and J. Polakis, "Navigating Murky Waters: Automated Browser Feature Testing for Uncovering Tracking Vectors," in *Network and Distributed System Security Symposium*, 2023.

[82] S. Koch, M. Wessels, B. Altpeter, M. Olvermann, and M. Johns, "Keeping Privacy Labels Honest," in *Privacy Enhancing Technologies Symposium*, 2022.

[83] T. T. Nguyen, M. Backes, and B. Stock, "Freely Given Consent? Studying Consent Notice of Third-Party Tracking and Its Violations of GDPR in Android Apps," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[84] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share First, Ask Later (or Never?) Studying Violations of GDPR's Explicit Consent in Android Apps," in *USENIX Security Symposium*, 2021.

[85] K. Kollnig, A. Shuba, R. Binns, M. Van Kleek, and N. Shadbolt, "Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps," in *Privacy Enhancing Technologies Symposium*, 2022.

[86] K. Kollnig, R. Binns, P. Dewitte, M. v. Kleek, G. Wang, D. Omeiza, H. Webb, and N. Shadboldt, "A fait accompli? an empirical study into the absence of consent to third-party tracking in android apps," in *Symposium on Usable Privacy and Security*, 2021.

A Full Static Measurement Results

Figure 5 provides an overview of the results of our static measurement.

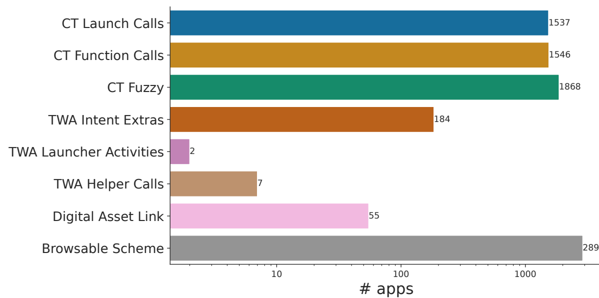

Figure 5: Frequency of CT, TWA, DALs, and browsable scheme app capabilities.

B Example DAL

Figure 6 shows a DAL that links the app with the ID `org.example.app` to the website example.org. This showcases DAL's well-defined structure and content for both the website and the app.

```
1  // in the app:
2  [{
3  "relation": [
4    "delegate_permission/common.handle_all_urls"
5    ],
6  "target": {
7    "namespace": "web",
8    "site": "https://example.org"
9  }}]
10 // on the web:
11 [{
12 "relation": [
13   "delegate_permission/common.handle_all_urls"
14   ],
15 "target" : {
16   "namespace": "android_app",
17   "package_name": "org.example.app",
18   "sha256_cert_fingerprints": [
19     "DE:CA:FB:AD:[...]:AA"
20     ]
21 }}]
```

Figure 6: An example DAL, linking *org.example.app* to *https://example.org*.

C Dynamic Analysis Cookie Notes

Listing 1 provides lists of cookie values we discarded manually, as described in Section 6.2.

```
'', 'A0', 'null', 'DOB=20240820', 'DE', '24', 'AF=NOFORM', '{}', 'v1%3B664x412x2
.625', '%5B%5D', 'en-US%2Fautodetect', '4', '1,en_US', 'Europe/Berlin', '5125',
'en', 'utmcsr=install|utmccn=(not set)|utmcmd=organic', 'T', 'v1%3B760x412x2
.625', 'C', 'CheckForPermission', '1159', '412|840|2.625', 'en_US', 'deleted',
'*.googleapis.com', '7!27021', '5', '%5B%D5', '3356', 'DOB=20240821', 'SRCHLANG=
en&THEME=1', '10', 'en-us', '5206', '11', '1154', 'GMT%2B0200', '3318', '"{}"',
'variant', 'none', 'full', 'false', '/', '1128', ' ', 'US', 'android', '412x760
', '2206', '*', '7!13564', 'NI', '3277', 'light', 'self', 'POST', '""', '04', '
e30=', '$38936', '5.2.2', '2', '3', '3220', 'USD', '1213', '"utmcsr=install|
utmccn=(not set)|utmcmd=organic"', '%2F', '1:', '2.625', '1:login|true:last_id
|11:', 'Disabled', 'SRCHLANG=en', '2024.23.0', 'None', 'ok', '1168', '5205',
'5212', 'on', '1115', 'desygner', '%5B%5BB%5D%5D', 'de', 'ENU', 'SRCHLANG=de', '
v=2&lang=en-us', 'true', 'A', '5121', '%7B%7D', 'GET', 'YES', 'en_us', 'EUR', '
production02', 'metric', 'en-US', '10.3.0', 'delete', 'v1%3B685x412x2.625', '
estsfd', 'bf', 'SRCHLANG=en&THEME=1&DM=0&CW=360&CH=636&SCW=360&SCH=636&BRW=MM&
BRH=MM&DPR=2.6&UTC=120&CIBV=1.1801.0', '1', 'en-US/autodetect', '0', 'e30', '
DELETED', '06-Jan-2025', 'mobile', '0.001', '3385', 'utm_source%3Dexternal-apk
%26utm_campaign%3Dexternal%26utm_medium%3Dorganic', '/*', 'OPT_OUT', '1163', 'n0
', '1218', 'F=1', '5147'
```

Listing 1: Shared cookie values we manually identified as trivial and removed for further analysis.

D Android and Browser Versions

Table 5 shows the exact versions of Android flavors and Browsers used for our study in Section 3.

Table 5: Versions of browsers and Android builds used in our capability study.

Device	OS Build	Chrome Version
Google Pixel 6a	AP2A.240605.024	126.0.6478.122
Pixel 6a w/ GrapheneOS	2024070201	⊘
Samsung Galaxy A13	TP1A.220624.014. A136BXXU2BVK3	126.0.6478.122
Huawei P40 Lite	12.0.0.295 (C432E5R6P2)	112.0.5616.48
Xiaomi Poco F3	14.0.7.0 (TKHEUXM) (TKQ1.220829.002)	126.0.6478.122
Oneplus Nord	CPH2409_11_C.26	126.0.6478.122

Browser	Version
Opera	83.3.4388.80648
Firefox	127.0.2
Tor Browser	14.0.2
UC Browser	13.8.4.1325
Brave	1.73.89
Vanadium	126.0.6478.122
Samsung Internet	26.0.0.52
Huawei Browser	14.0.5.302
Mi Browser	13.16.1-gn