Bridges to Self: Silent Web-to-App Tracking on Mobile via Localhost

Tim Vlummens*
COSIC, KU Leuven

Aniketh Girish*
IMDEA Networks Institute

Nipuna Weerasekara
IMDEA Networks Institute

Frederik Zuiderveen Borgesius
Radboud University

Gunes Acar
Radboud University

Narseo Vallina-Rodriguez
IMDEA Networks Institute

Abstract

Modern browsers and mobile operating systems leverage sandboxing and process isolation to separate web and app contexts. However, in this paper, we show that these isolation guarantees can be — and had been — broken in practice on Android devices by Meta and Yandex to enable cross-context tracking that bridges web tracking with native identities.

Using a combination of large-scale web crawls from USA and EU vantage points and systematic Android app analysis, we characterize a previously undocumented family of web-to-app tracking paradigms that exploit web standards such as HTTP(S), WebSocket, and WebRTC to connect mobile and web contexts on localhost. By linking pseudonymous web cookies to long-lived native user IDs, these channels enable persistent and stealthy cross-context tracking, and de-anonymization. This new technique defeats protections such as cookie clearing, Incognito mode, Mobile Advertising ID (MAID) resets, VPNs, and Android's work/personal profile separations. We further show that Meta Pixel and Yandex Metrica initiated localhost bridging prior to accepting cookie consent banners. We evaluate browsers' patching efforts and defenses to these attacks in response to our responsible disclosure, and the upcoming Local Network Access (LNA) permission, which introduces user prompts for accessing localhost and local network addresses. In doing so, we identify additional side-channels that bypass such protections using (i) global-unicast IPv6 addresses in WebRTC; and (ii) mDNS lookups on `*.local` domains. Our results, together with an enclosed legal analysis, expose structural shortcomings and the need to revisit platforms' and browsers' isolation principles, threat and trust models, protocol standards, and app review processes to prevent future cross-context abuse.

1 Introduction

Many security and privacy protections in web browsers and mobile operating systems rely on a foundational principle: isolation. This principle limits how untrusted code can access sensitive data or interfere with other processes, preserving confidentiality and integrity [86].

On the web, browsers enforce isolation primarily through origin-based policies such as the Same-Origin Policy [123], which prevent scripts from different origins from directly accessing each other's data or execution contexts. Modern browsers have further extended isolation by partitioning client-side storage on a per-site basis to prevent cross-site tracking [92]. Mobile operating systems adopt process isolation where each application gets assigned a distinct Linux user ID, running in a siloed sandbox [24, 86]. The sandbox prevents direct access to other apps' code, memory, and storage, forcing all cross-app interactions through well-defined Inter-Process Communication (IPC) channels [86]. This strong isolation is a key reason why Android and iOS are widely considered resilient to direct cross-app data leakage.

Prior research has extensively studied privacy and tracking abuses on both web and mobile ecosystems [13, 49, 84, 113, 114], but largely as disjoint problems, implicitly assuming that web and native apps are isolated. This paper shows that this assumption of isolation fails by uncovering a deterministic tracking mechanism where native apps and in-browser scripts coordinate via localhost channels — using HTTP(S), WebSockets, or WebRTC — to link web and native identities. Under current web standards, however, localhost is classified as a *potentially trustworthy origin* [121], which facilitates such abuse by exempting it from mixed-content restrictions.

As opposed to websites, many native apps require users to log in and provide their identities through their real name or email address, or they can programmatically access unique device IDs such as the Android Advertising ID (AAID) [67]. Therefore, unvetted localhost channels enable the direct coupling of web cookies with mobile app identifiers and users' identity, effectively de-anonymizing users' web activities.

This emerging web-to-app ID sharing paradigm bypasses standard browser and mobile privacy protections, including sandboxing, cookie clearing, or Incognito Mode, and potentially violates mobile platform policies. Furthermore, it en-

*The two lead authors contributed equally to this work.

ables malicious apps to eavesdrop on users' web activity.

Through systematic static and dynamic code analysis and web crawls, we show that Meta and Yandex have deployed this novel and stealthy tracking method for years, escaping browser and mobile platform security mechanisms. Specifically, our analysis reveals that native Android apps — including Facebook, Instagram, and several Yandex applications — covertly listen on local ports to receive user identifiers and cookies from Meta Pixel and Yandex Metrica scripts embedded on millions of websites. When loaded in users' Android browsers, these scripts establish silent connections with the corresponding native apps via localhost sockets, enabling deterministic and persistent cross-platform tracking of hundreds of millions of Android users globally. Our study makes the following contributions:

- We study localhost tracking, a novel tracking technique that enables the linking of web and mobile app identities for deterministic tracking purposes using unrestricted access to localhost channels.
- Through large-scale web crawls and mobile app analysis, we measure this tracking mechanism's prevalence. We find that it is used by Meta and Yandex, two companies whose mobile apps used by billions of users, and whose advertising/analytics scripts are present on millions of websites.
- We develop and disclose similar, novel attack vectors that employ mDNS lookups, and global unicast IPv6 candidates in WebRTC.
- We responsibly disclose our findings to browser and mobile platform vendors (§11), which resulted in mitigations already deployed and shipped to end users. Yandex and Meta terminated the tracking method on June 3rd 2025, on the day of our public disclosure.

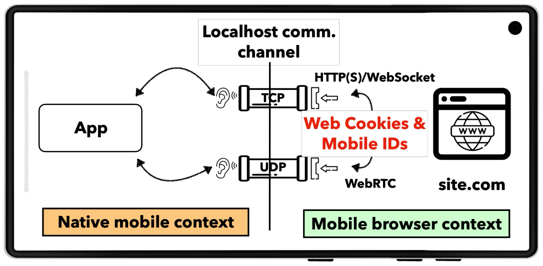


Figure 1: Localhost covert tracking threat model. The website connects to an app running on the same device using localhost communication channels such as HTTP(S), WebSocket or WebRTC. The website may pass a unique user ID, which is then linked to user's persistent mobile app account or identity.

2 Background

This section reviews web privacy controls (§2.1), mobile privacy controls (§2.2), and introduces technical concepts related to mDNS and WebRTC (§2.3), two methods that can be used to bypass browser and platform isolations.

2.1 Web Privacy Controls

Browsers enforce the same-origin policy to isolate content from different domains, and they limit cross-site tracking via third-party cookie blocking and state/storage partitioning [88, 90, 92, 93]. Modern browser-based privacy features further restrict cross-site data flows [102, 129]. However, these defenses typically assume that web content does not directly interact with native applications.

Web standards treat localhost as a *potentially trustworthy origin* [128], despite not being a secure or encrypted context [91]. Unrestricted access to localhost and local networks remains a persistent security gap in browsers, enabling vulnerabilities such as DNS rebinding [78]; remote code execution (RCE) and unauthenticated access in routers [100, 101] and IoT devices [14]; password managers [72]; video conference software [32]; and local ML model servers [108].

2.2 Mobile Privacy Controls

UNIX-based mobile operating systems (OSes) such as Android and iOS enforce process sandboxing [24]. Yet, platform-specific design choices determine how easily data can be accessed by processes and how it moves across contexts (e.g., app↔app, or app↔web). iOS limits apps' ability to run in the background, suspending their execution unless they declare use of one of the platform-allowed background modes (e.g., audio play, VoIP) [27]. In contrast, Android is more permissive, allowing services and broadcast receivers to run background tasks [19, 20].

Inter-process Communication (IPC). Android exposes various IPC mechanisms (e.g., intents, content providers, and broadcasts) for cross-context communications [23]. Developers may protect such flows through custom permissions [56]. Cross-app communication on iOS is intentionally limited to URL schemes/Universal Links or app groups (i.e., shared containers for the same developer) [26, 30].

Permission Model. Modern mobile platforms regulate access to sensitive capabilities and data through a permission model, requiring explicit user consent to access features such as location, microphone/camera, and Bluetooth scanning [22, 58]. To enhance user awareness and transparency, Android's Privacy Dashboard (Android 12+) shows recent app permission accesses [18]. In iOS, accessing special restricted privileges such as local-network discovery requires additional *entitlements* that are approved through app reviews [28]. The Android Advertising ID (AAID), a resettable ID relevant to our threat model described in §3, is commonly used for advertising and tracking. On Android 13 (API 33) and later, accessing the ID requires declaring the install-time `AD_ID` permission;

otherwise, it is zeroed.

Platform Policies and Vetting Processes. Mobile platforms define policies, and use app review processes to restrict the distribution of potentially harmful, insecure or privacy-violating apps [125]. For example, these policies restrict cross-app tracking and ID bridging [65, 66], and prohibit the distribution of apps that "interfere with, disrupt, damage, or access in an unauthorized manner the user's device", including other apps running on the device [71]. Apple's App Tracking Transparency (ATT) requires opt-in consent for tracking and forbids fingerprinting [25]. Google Play's Data Safety section requires developers to disclose how they collect, use, and share user data, including data shared with third parties [70]. Prior research found that the effectiveness of vetting processes for privacy policy enforcement is limited in practice [55, 103, 113, 114, 125, 126, 131, 133].

2.3 WebRTC and Multicast DNS (mDNS)

Web Real-Time Communication (WebRTC). Modern browsers support real-time, peer-to-peer (P2P) audio/video, and data transmissions though WebRTC protocols, which are exposed as JavaScript APIs. WebRTC negotiates connectivity, and discovers paths between peers using the standardized Interactive Connectivity Establishment (ICE) protocol [76]. ICE collects three candidate types that indicate protocols and routing needed for connectivity: (i) host candidates derived from local network interfaces; (ii) server-reflexive candidates, in the form of IP:PORT pairs, learned via Session Traversal Utilities for NAT (STUN);[1] and (iii) relayed candidates allocated by Traversal Using Relays around NAT (TURN) servers when direct connectivity fails.[2] ICE packages these candidates along with codec and transport parameters into Session Description Protocol (SDP) offer/answer exchanges to establish a P2P connection. WebRTC specifications treat browser-generated SDP as an opaque artifact that must be passed unchanged, requiring negotiation to occur exclusively through standardized APIs rather than direct SDP modification (*"SDP munging"*) [73, 77]. SDP munging is discouraged by both IETF and WebRTC standards, to avoid compatibility issues and unintended data leakage [124].

Multicast DNS (mDNS). The mDNS protocol provides DNS-like name resolution on the local network address space without a unicast DNS server. Hosts advertise and resolve `.local` names via multicast on UDP:5353 (IPv4 `224.0.0.251`, IPv6 `ff02::fb`) [42]. Paired with DNS-Based Service Discovery (DNS-SD), mDNS enables zero-configuration discovery of nearby services (*e.g.,* printers, smart devices) by publishing service types and attributes [41]. Because mDNS announces the presence of the device and may propagate device metadata, it can reveal local topology and enables household fingerprinting or cross-device tracking [57]. Mobile platforms expose APIs for mDNS/multicast discovery, but their use is subject to platform controls such as iOS Local Network prompts since iOS 14 or the experimental Android's Local Network permission since version 16 [21, 29].

[1]STUN is a lightweight protocol that reveals an endpoint's public-facing address and assists with NAT traversal checks.

[2]TURN relays data through a server when firewalls or NATs block direct routes.

3 Threat Model

This paper introduces and demonstrates a novel tracking method that exploits unrestricted access to localhost sockets on mobile and websites to bridge web and native tracking identifiers. Figure 1 provides a high-level description of this attack and its implications.

The method requires the attacker to control a native mobile app, and a website or third-party scripts embedded on other websites. Both Android and iOS allow user-space apps with installation-time permissions or entitlements (e.g., `INTERNET` for Android) to open a listening or sending socket on the loopback interface. Until the deployment of LNA in October 2025, browsers allowed localhost access through JavaScript APIs without user or platform mediation. As a result — despite operating with limited user-space privileges and process sandboxing constraints — the attacker can establish via HTTP(S) or WebRTC communication channels between web scripts and other processes on the same device.

By bridging web and native contexts to link ephemeral web IDs to long-lived mobile app IDs, the method enables persistent cross-platform tracking. This web-to-mobile bridging channel (i) breaks fundamental mobile and web isolation safeguards such as OS sandboxing, Incognito mode [69], and cross-origin access restrictions; (ii) defeats privacy-enhancing tools such as VPNs; (iii) defeats Android's work-personal profile separations; and (iv) enables persistent reconstruction of mobile advertising IDs and cookies after reset events, defeating the purpose of resettable IDs and cookie deletion [68].

For achieving global cross-platform tracking, the attacker must gain access — directly as developer or through partnerships with other firms — to (i) a prevalent web script; and (ii) a widely installed app, so it is likely that both the attacker's script and their app are running on the same device. The attack can be launched by the attacker's third-party advertising and analytics scripts, and can even be micro-targeted to specific users through Real-Time Bidding (RTB) processes. The attack generalizes to other types of software, including browser extensions, native SDKs and in-app browsing technologies such as WebViews and CustomTabs [33, 131].

3.1 Attack Feasibility

Android. We develop two Proof-of-Concept (PoC) Android apps to demonstrate the feasibility of the attack in different mobile browsers, both on Android 15 and 16. The PoC apps include (1) an HTTP request/UDP payload logger, and (2) an mDNS lookup logger. Both apps bind to the localhost address and listen on an arbitrary port for incoming requests, allowing us to evaluate whether HTTP(S), WebSocket, and WebRTC connections on localhost are permitted between web and native contexts. We find that all major browsers permit some form of localhost communications between web scripts and native mobile apps (Table 3). Up to and including Android 15, none of the PoC apps required user permission. On Android 16, listening to mDNS lookups requires the experimental Local Network permission [21]. However, the scope and semantics of local network IP addresses protected by this permission differ from those of localhost IPs, hence potentially leading to user confusion. For these reasons, we argue that localhost access should be handled separately from local network access controls. §6 presents a detailed evaluation of different browsers and applicable mitigations against real-world instantiations of the localhost attack.

Android Enterprise Profiles. Android supports Work Profiles for managing work-related apps and data, ensuring that personal and work data are kept separate [17]. We demonstrate through our PoC apps that the localhost attack remains feasible across profiles without any restrictions based on profile scope (e.g., Work or Personal), defeating the security assumptions and expectations of enterprise network management. The attack remains feasible even with active VPN tunnels.

iOS. We empirically verify the feasibility of this attack on an iPhone 14 Pro running iOS 18.5. On both Safari (v18) and Brave (v1.78.1), we successfully fetch web resources from a native HTTP server actively listening on localhost on the phone. Concretely, we run a local server iOS app (PocketServer) on a specific port, and load a resource from that particular `localhost:port` on a public HTTPS test page. Practical exploitation on iOS depends on the ability of a malicious app to remain active in the background and on Apple's ability to effectively detect deceptive background executions. Although iOS imposes more restrictive controls than Android over background executions [27], apps could run local servers in the background if they receive the necessary (background) entitlement. Indeed, the iOS app we use in the tests (PocketServer) could run in the background without interruption, using the `NoIdleSleepAssertion` by claiming to play audio despite having no UI indicator.

4 Methodology

Detecting and characterizing localhost web-mobile bridging abuse in the wild requires joint analysis of both websites and mobile applications, as the attack requires their coordination. This requirement significantly increases analysis and measurement complexity, as it entails reasoning over a large combinatorial space of website-app pairs (§4.1). We address this challenge by adopting a principled methodology that combines instrumented web crawling (§4.2) with static and dynamic Android app analysis techniques (§4.3). This approach allows us to efficiently reduce the problem space and identify candidate website-app pairs for in-depth analysis.

4.1 Dataset

Websites. We crawl the top-100,000 sites from the February 2025 CrUX rankings [47, 64] (February 2025). We chose the CrUX ranking following recent best practices [115], as it represents the set of web-specific domains, grouped into popularity tiers, more accurately than traditional top lists. We additionally use historical HTTP Archive data to longitudinally trace the emergence and evolution of web-to-app bridges over time, using fetched resources, invoked web APIs, and error logs recorded in the archives [75].

Mobile Apps. We construct a corpus of 5,000 apps from Google Play's Top Free charts in the EU and USA using the open-source `google-play-scraper` [107]. We fetch (i) the overall Top Free list and (ii) Top Free apps per category, including store metadata (*e.g.,* package name, install counts). We then rank the top-5k apps by cumulative install counts, as it is a more stable signal of app popularity compared to active installs. We use Google Play (on-device) with the app seed list to automatically download and run apps onto a pool of instrumented Pixel devices located in the EU.

4.2 Web Analysis

We perform web crawls using a customized version of DuckDuckGo's Tracker Radar Collector (TRC) [46], an open-source Puppeteer-based crawler equipped with anti-bot measures (e.g., hiding 'navigator.webdriver'). We leverage TRC's *collectors* to capture HTTP requests, cookies, and accesses to certain JavaScript API methods and properties. We crawl the CrUX top-100k sites using machines located in the USA (New York) and the EU (Frankfurt), customizing the `User-Agent` string and viewport to emulate an Android phone (Google Pixel 7 with Android 13), an iPhone (iOS version 18), and a Windows 10 computer.

We modify TRC's request collector to record WebSocket frames in addition to HTTP(S) requests, and we monitor WebRTC usage by instrumenting selected WebRTC-related JavaScript API methods (*Appendix A*). After each crawl completes, we process the collected data using a dedicated analysis script to identify potential localhost communications. Specifically, we check if HTTP request URLs (including WebSocket handshakes) or the arguments passed to the *setLocalDescription* and *setRemoteDescription* WebRTC methods contain either "localhost", "0.0.0.0", "::1" or a "127.0.0.0/8" addresses, alongside any ports. We additionally resolve the domain names of the request URLs to identify cases that map to the localhost address. We do not consider WebRTC candidates for these lookups, as they only contain IP or mDNS (`.local`) addresses. While our localhost definition aligns with the IP ranges considered by the Local Network Access specifications (LNA) [122], we also include requests to `0.0.0.0`, which has been abused in prior real-world exploits [108].

We modify the crawler to automatically accept cookies using Priv-Accept [79] to simulate user consent. After a site has loaded, the crawler accepts all cookies when a site-provided consent form is rendered, and it then waits ten seconds without further user interaction while collecting all network traffic. In total we perform four independent crawls per location where cookies are accepted: two before and two after our public disclosure. These crawls include a mobile (Android) and desktop (Windows 10) crawl in April and May 2025. Following our public disclosure in June 2025, we perform an Android and an iOS crawl to ensure the localhost tracking method is not used anymore. We perform an additional *no-consent* crawl on sites where we observed localhost communications in our initial (April 2025) Android crawl to simulate whether user consent is necessary for localhost tracking. In this crawl, the crawler does not interact with consent forms or any other elements on the page. Across all runs, our crawler successfully visited 92–96% of the 100K sites.

4.3 App Analysis

To detect and validate localhost-based web–to–app or web–to–SDK channels, we combine static and dynamic analysis as neither approach is sufficient in isolation. Static analysis produces false positives (e.g., localhost use for debugging or IPC) and false negatives (e.g., due to code obfuscation or dynamically-loaded code) when scanning for loopback address usage, embedded HTTP servers, STUN/TURN channels, or in-app listeners, while dynamic analysis alone provides partial visibility because behaviors may depend on timing conditions or server-side commands. This multi-modal approach is essential to balance scalability and visibility; accordingly, we rely on dynamic analysis as the primary technique to obtain concrete evidence of abuse, using static analysis to interpret and contextualize observed behaviors, consistent with prior large-scale mobile app measurement studies [33, 57, 58, 81, 83, 84, 113, 117, 131].

Using the Android app dataset (§4.1), we automatically test each app in a pool of OS-instrumented Pixel smartphones (6a/3a; Android 12-16), providing visibility into runtime resource access, including permission-protected APIs, file I/O, process lifecycle events, and socket creation and binding. We use in-process monitoring using Frida [111] to observe and modify specific functions in both Java and native code to (i) bypass proprietary certificate pinning and custom TLS verification [110]; (ii) trace crypto operations; and (iii) record variable assignments and serialized fields immediately before they are written to sockets. In parallel, we use (i) `tcpdump` and `mitmproxy` to trace browser JavaScript requests to localhost and the corresponding responses from native apps; and (ii) `netstat` to identify open ports.[3] The combination of these techniques enables us to capture browser JavaScript-generated requests to localhost and app responses, including any immediate app- or web-to-cloud data dissemination.

To trigger web-to-app flows, we load pages found in our active crawls that embed target scripts (e.g., Meta Pixel, Yandex Metrica) in major mobile browsers. Each app is automatically executed for 10 minutes using Android Monkey to generate synthetic UI interactions. To pass SSO-based registration walls and increase code-path coverage, devices are provisioned with pseudonymous phone numbers, email addresses, and usernames. Because input identifier values are known per device, we automatically search for direct occurrences of these values in captured traffic, including common transformations (e.g., hashes such as MD5, SHA-1, and SHA-256).

4.4 Limitations

Detecting a complex, cross-platform abuse such as localhost-based tracking poses inherent scalability and observability challenges: certain web-to-app localhost flows can only be triggered by testing specific app-website combinations, yet exhaustively evaluating all possible pairs across app stores and the web is technically infeasible. We partially mitigate these factors through efficient triage using large-scale web crawls, on-device dynamic instrumentation, and static/decompiled code analysis. Moreover, as §5 shows, real-world instantiations of this abuse implement evasive features relying on remote configuration, execution delays, and code obfuscation. As a result, we do not claim completeness, but rather a demonstration of both the technical feasibility of the attack and its real-world exploitation, while providing concrete evidence to inform the design of mitigations that can benefit billions of users worldwide.

While our web crawls emulate Android, desktop and iOS browsers, our app analysis is Android-focused. In our measurements and manual analyses, Meta's and Yandex's scripts trigger requests only on mobile Android browsers, as determined by user-agent-based checks. No localhost usage is observed in iOS and desktop via targeted small scale on these platforms. In §3.1 we show how localhost ID-sharing is also technically possible in iOS, though stricter background-execution and process controls make it more challenging.

5 Empirical Results

From the web and Android app analysis methodologies described in §4, we identify two large-scale deployments of localhost-based tracking attributable to Yandex (§5.1) and Meta (§5.2). §5.3 quantifies the prevalence of websites embedding Meta's and Yandex's scripts and characterizes their

[3]We attribute open ports to apps by resolving PID/UID to package names.

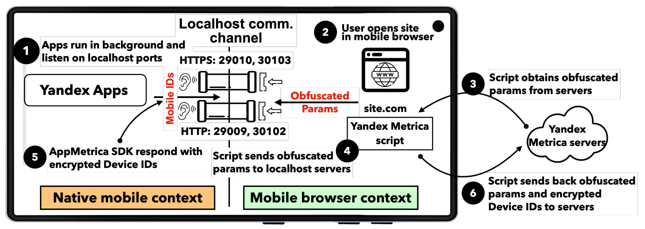

Localhost comm. channel
1 Apps run in background and listen on localhost ports
HTTPS: 29010, 30103
2 User opens site in mobile browser
Yandex Apps
Mobile IDs
Obfuscated Params
site.com
3 Script obtains obfuscated params from servers
Yandex Metrica servers
Yandex Metrica script
HTTP: 29009, 30102
5 AppMetrica SDK respond with encrypted Device IDs
Script sends obfuscated params to localhost servers 4
6 Script sends back obfuscated params and encrypted Device IDs to servers
Native mobile context
Mobile browser context

Figure 2: Diagram showing the steps of Yandex Metrica's localhost communication.

consent-handling behavior. §5.4 discusses other, potentially legitimate, uses of web-to-mobile localhost channels.

5.1 Yandex Metrica

Yandex Metrica [135] and Yandex AppMetrica are Yandex's web and mobile analytics solutions, respectively [134]. Analyzing our web crawl results, we observe that the Yandex Metrica and AppMetrica products exchange tracking identifiers via localhost channels for web-to-mobile matching. Specifically, Yandex Metrica's JavaScript initiates HTTP and HTTPS requests with opaque parameters to localhost over a set of pre-defined TCP ports: i.e., 29009, 29010, 30102, and 30103. Several native Android Yandex-owned apps such as Yandex Maps and Navigator, Yandex Search, and Yandex Browser implement background localhost HTTP servers that actively listen for incoming connections on these ports. Table 5, in Appendix B, lists the Yandex-owned apps that we find listening on localhost ports using dynamic instrumentation, along with their package names and versions. Code inspection indicates that this behavior is tied to the AppMetrica SDK embedded in these apps rather than to app-specific logic.

A targeted analysis of Yandex Maps and Navigator (`ru.yandex.yandexnavi`) shows that the localhost server functionality disappears in the period surrounding our public disclosure, between late May and early June 2025.

5.1.1 Communication Flow

Yandex native apps contact a Yandex-controlled endpoint (*startup.mobile.yandex.net*) to retrieve a runtime configuration that specifies the local ports to listen on (e.g., 30102, 29009). This server-controlled mechanism allows the localhost channel configuration to be updated dynamically without app updates, resembling command-and-control behaviors and making it robust against static blocklisting and filters. Figure 2 shows the data flow of Yandex's implementation:

❶ Following an initial delay, the app launches a background HTTP(S) server that listens on server-defined ports. ❷ The user visits a website embedding the Yandex Metrica JavaScript. ❸ The script contacts Yandex's backend (*mc.yandex.com*) to fetch a set of obfuscated parameters, as shown in Listing 3. These parameters include a validation token (`t`), ephemeral server-generated values (`a` and `b`), and a tag identifier (`c`) corresponding to the website's Metrica ID. ❹ The script forwards these parameters to the native localhost server over HTTP and HTTPS. Upon receiving the request, the mobile app validates the `t` token and constructs a response parameter that contains the web-supplied parameters and native identifiers, including the Android Advertising ID (AAID) [67], an AppMetrica SDK-specific UUID, and additional device IDs accessible via Java APIs (Figure 3). The response parameter is encrypted using an AES-RSA hybrid scheme and Base64-encoded. ❺ The response parameter is returned to the requesting JavaScript code running on the browser context. ❻ The Metrica JavaScript immediately relays the response parameter to Yandex's backend servers (e.g., *mc.yango.com*), enabling the matching of web traffic activity to users' mobile app identities.

Original payload

```
eyJ1dWlkIjoiOGMyZjRiMGQxNDMyNGFlNWIwOTg0NDB
mZDYwYzE5ZjkiLCJkZXZpY2VfaWQiOiJh
MmI5YmU2MWQ3MTc5ODU5ZTRiZWIxZTY0Y
2U0ZmFmNSIsImdvb2dsZV9haWQiOiJhODU3MWRhMC03
ZDNlLTRmMTItOGNjNC05MDMwNTU5ZDI5ODgifQ==
```

Decoded payload

```
{ "uuid": "8c2f4b0d14324ae5b098440fd60c19f9",
  "device_id": "a2b9be61d7179859e4beb1e64ce4faf5",
  "google_aid": "a8571da0-7d3e-4f12-8cc4-9030559
      d2988"}
```

Figure 3: The top shows the encrypted payload. The corresponding decoded plaintext JSON payload containing device identifiers before encryption is shown at the bottom.

To validate the content of the response, we use static analysis to locate the specific code segment responsible for assembling the localhost response. We then hook these methods with Frida to intercept the plaintext JSON payload immediately before encryption. Table 6 in Appendix D lists additional Yandex domains and collected data types observed during our analyses, such as MAC addresses and precise geolocation.

Temporal Evolution. According to data from the HTTP Archive's historical crawls [75], Yandex has used localhost channels since February 2017, initially using only HTTP requests to TCP ports 29009 and 30102. In May 2018, they added support for HTTPS on TCP ports 29010 and 30103, as Table 1 reports. Both methods were used simultaneously for over eight years until June 3rd 2025, when Yandex discontinued the practice after our public disclosure.

Table 1: History of the used methods based on historical HTTP Archive data. * Meta Pixel script was last seen sending via HTTP in Oct 2024, but Facebook and Instagram apps still listened on this port. They also listen on port 12388 for HTTP, but we have not found any script sending to 12388. ** Meta Pixel script sends to these ports, but Meta apps do not listen on them. We speculate that this behavior could be due to slow/gradual app rollout.

	Method	Start date (first seen)	End date (last seen)	Ports
Yandex	HTTP	Feb 2017	Jun 2025	29009, 30102
	HTTPS	May 2018	Jun 2025	29010, 30103
Meta	HTTP	Sep 2024	Oct 2024*	12387
	Websocket	Nov 2024	Jan 2025	12387
	WebRTC STUN (w/ SDP Munging)	Nov 2024	Jun 2025	12580–12585
	WebRTC TURN** (w/o SDP Munging)	May 2025	Jun 2025	12586–12591

5.1.2 Use of Evasion Techniques

Yandex Metrica implements features that can evade detection using conventional program analysis methods:

JavaScript and Java Code Obfuscation. On the JavaScript side, Yandex encodes parameters (e.g., `t`, `a`, `b`, and `c`) and the mechanisms used to dynamically retrieve localhost configuration from remote endpoints, limiting visibility into how these ephemeral keys are generated and how they influence the observed behaviors. The obfuscated nature of the Yandex Metrica script further prevents us from determining the specific events that trigger localhost traffic on the browser side, but manual testing reveals that loading the script is sufficient to initiate the communication (§5.3). On the native side, we observe pervasive class and method renaming, synthetic wrappers, and convoluted control structures (e.g., deep nesting and excessive try/catch blocks) that obscure the logic of the localhost server and parameter validation. To overcome these challenges, we combine static inspection with Frida-based runtime instrumentation to recover the plaintext JSON containing AAID and other device-level IDs before encryption.

Delayed Execution. Yandex Metrica's runtime configuration—retrieved from *startup.mobile.yandex.net*—includes a `first_delay_seconds` parameter that defers the activation of its localhost server listeners for roughly three days. Consequently, local server creation is unlikely to be detected by standard dynamic analysis pipelines, which usually test apps for a limited period of time on the order of minutes or hours, rather than days [58, 84, 103, 112, 113].

Dynamic Port Selection. Because Yandex Metrica retrieves localhost ports at startup, they can be changed dynamically via server-side configuration without requiring an app update, enabling rapid evasion of port-based blocking.

HTTPS Connection to a Domain Resolving to Loopback. Yandex scripts send HTTPS requests to the *yandexmetrica.com* domain, which resolves to `127.0.0.1`. The use of a domain, rather than an IP address or the *localhost* hostname, makes the abuse less detectable as it appears as normal remote traffic. App analysis reveals that Yandex bundles a complete TLS certificate for *yandexmetrica.com* directly inside their private storage in a PKCS#12 file protected by a fixed password hardcoded in the app. This includes the matching private key along with a full certificate chain signed by GlobalSign, a trusted CA. The Certificate Transparency (CT) logs confirm that this certificate has been registered since May 2018 [4]. Because this certificate is valid and publicly trusted, browsers on the device can establish encrypted connections to the app's localhost service without triggering any certificate warnings, as if it were a normal and legitimate remote server. We note that the private key for the yandexmetrica.com certificate can be extracted by an attacker to perform spoofing and TLS interception attacks with basic code analysis techniques.

5.1.3 Additional Risks: Browsing History Leak

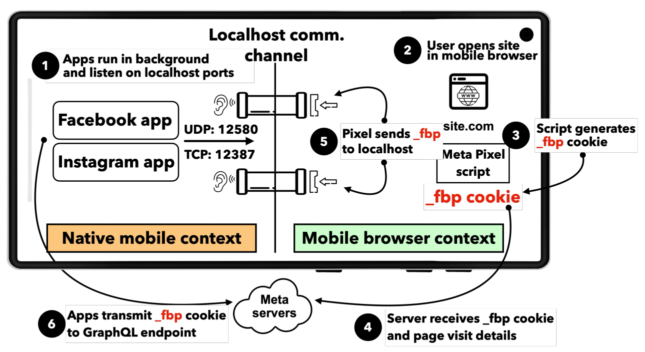
Using unencrypted HTTP communication for web-to-native ID sharing may expose users' browsing history to third-party apps that also listen on the aforementioned ports, inferring users' browsing history by parsing the `Origin` HTTP header. This browsing activity disclosure also affects the first implementation of Meta's localhost feature, as we present in §5.2.1. We developed a PoC app to demonstrate how a malicious actor could partially harvest users' browsing history, shown in Fig. 7 in Appendix C.

5.2 Meta Pixel

The Meta Pixel is a tracker script that is used on millions of websites for advertising and marketing purposes [97], including re-targeting ads to web visitors on platforms such as Facebook and Instagram. According to the HTTP Archive, Meta has deployed this web-to-mobile bridging mechanism since late 2024, evolving its implementation and use of network protocols over time as we detail in §5.2.1.

When Android users visit a website with the Meta Pixel, it sends the first-party `_fbp` [96] cookie to the Facebook (e.g., v515.0.0.23.90) and Instagram (e.g., v382.0.0.43.84) apps using WebRTC on ports UDP:12580-12585. The Threads app (v377.0.0.0.30) also contains the logic to listen on these ports, although we do not observe this behavior at runtime. We find

[4]https://crt.sh/?q=yandexmetrica.com



Localhost comm. channel
❶ Apps run in background and listen on localhost ports
❷ User opens site in mobile browser
Facebook app
Instagram app
UDP: 12580
TCP: 12387
❺ Pixel sends _fbp to localhost
site.com
❸ Script generates _fbp cookie
Meta Pixel script
_fbp cookie
Native mobile context
Mobile browser context
Meta servers
❻ Apps transmit _fbp cookie to GraphQL endpoint
❹ Server receives _fbp cookie and page visit details

Figure 4: Diagram showing Meta Pixel's localhost operations.

no evidence of code implementing localhost capabilities on WhatsApp. The `_fbp` cookie is the third most common first-party web cookie, present on approximately 25% of the top 1-M websites, according to Web Almanac 2024 [74].

Meta's Cookies Policy [99] states that the `_fbp` cookie is a first-party cookie that *"identifies browsers for the purposes of providing advertising and site analytics services and has a lifespan of 90 days"*. Technically, a first-party cookie cannot be used for cross-site tracking, as it is scoped to the embedding website's (first-party) domain. In addition to users' `_fbp` cookie, the Meta Pixel also collects data such as the visited URL, pixel ID, page details, and *events* such as *Add to cart*, *Search* and *Checkout*, along with relevant metadata when applicable (e.g., product name and price). Publishers can customize Meta Pixel to collect additional data, including email hashes from registration forms [94].

Unless a user is logged into Facebook or Instagram on their mobile browsers [5], Meta cannot directly link their Pixel/web activities to their Facebook or Instagram accounts. The localhost bridge circumvents this limitation, allowing Meta to persistently and deterministically link user identities and activities across web and native contexts, despite Incognito mode, VPNs, or clearing cookies.

5.2.1 Communication Flow

Figure 4 represents the observed data flow of Meta's latest localhost tracking implementation. ❶ When users open the native Facebook or Instagram app, a background service that listens on UDP:12580–12585 (used for STUN messages) and on TCP:12387–12388 is started. [6] ❷ When a user visits a webpage with the Meta Pixel, ❸ the script generates and ❹ uploads the `_fbp` cookie to the host `https://www.facebook.com/tr` along with metadata such as the page URL, browser metadata, and event type (e.g., PageView, AddToCart) , ❺ The script uses SDP munging to insert the `_fbp` cookie into the SDP `ice-ufrag` field of a WebRTC offer (§2.3). This causes the browser to emit a STUN Binding Request to the loopback address (127.0.0.1) on UDP:12580–12585. ❻ The native Meta apps parse the STUN Binding Request and extract the `_fbp` value from the `USERNAME` attribute, later forwarding it through a GraphQL mutation to Meta servers on `https://graph.facebook.com/graphql`. As illustrated in Figure 5, the mutation includes fields containing the *_fbp* cookie value: `conversion_id=<fbp-value>` and `conversion_id_type=DOMAIN_AND_FBP`. Finally, the app replies on localhost with a STUN Binding Success Response that includes message integrity and fingerprint attributes. When Meta Pixel receives this response, it records a timestamp *(aidTime)* in localStorage. As a client-side rate-limiting measure, the script refrains from sending further traffic to localhost for roughly one hour after the stored timestamp.

Temporal Evolution. HTTP Archive's monthly crawls, summarized in Table 1, indicate that Meta's use of localhost implementation evolved quickly over roughly nine months since September 2024 until the 3rd of June 2025, when it was discontinued on the day of our public disclosure. The initial deployment used plain HTTP requests to TCP port 12387, starting in September 2024. In November 2024, Meta switched to WebSocket traffic on the same port and, in parallel, introduced WebRTC STUN binding requests across a UDP port range of 12580-12585, injecting the `_fbp` cookie into the SDP offer. In May 2025, following our first responsible disclosure, Chrome engineers announced that the specific SDP-munging technique that Meta used will be disabled to prevent abuse [16]. A few weeks after the announcement, Meta Pixel switched to using the WebRTC TURN protocol [130] on localhost ports UDP:12586–12591 to send the `_fbp` cookie.

Global WebRTC Usage Surge due to Meta Pixel. The widespread deployment of the Meta Pixel and its reliance on WebRTC APIs can be indirectly observed in trends reported by Chrome's Platform Status [60], which show a significant increase in WebRTC API usage across popular sites. Figure 6 highlights this trend for WebRTC's `RTCPeerConnection.SetLocalDescription` function, one of the methods invoked by Meta Pixel's WebRTC implementation. Note the surge in usage overlaps with the active period of Meta's WebRTC usage between November 2024 and June 2025.

Background Behavior. The Facebook and Instagram apps start the listening servers a few tens of seconds after being moved to the background. As the browser is in the foreground when users navigate the web, this does not hinder localhost communications. Instead, it makes it harder to capture evidence during app testing, as automated testing typically studies apps in isolation and while executed in the foreground.

Transparency. We found no public documentation describing Meta's localhost capabilities. In September 2024, when a number of web developers reported on Meta's own developer forums mysterious localhost requests triggered by Meta Pixel, their questions were left unanswered [95, 98]. The de-

[5]This is likely very uncommon given the availability of native Android apps offering better performance and more features.

[6]When both apps are running simultaneously, they attempt to bind sequentially and will occupy the first available port.

```
POST https://graph.facebook.com/graphql HTTP/2.0
x-fb-network-properties: Validated;dhcpServerAddr=172.16.4.1;LocalAddrs=/fe80::74...
x-fb-friendly-name: FBGraphQLOnDeviceConversionIDMutation
...
method=post&pretty=false&format=json&server_timestamps=true&locale=en_US&fb_api_req_friendly_name=
    FBGraphQLOnDeviceConversionIDMutation&fb_api_caller_class=graphservice&client_doc_id=1489694xxx...&
    fb_api_client_context={"is_background":true}&variables={"request":{"conversion_id_type":"DOMAIN_AND_FBP",
    "conversion_id":"fb+1+17xxx282xxx+xx136xxxx3xx"}}...
```

Figure 5: Details of the request that Meta apps use to upload the _fbp cookie to Meta servers, as a GraphQL mutation named `FBGraphQLOnDeviceConversionIDMutation`. In the payload, conversion_id is set to the browser's _fbp cookie; and conversion_id_type is `DOMAIN_AND_FBP`.

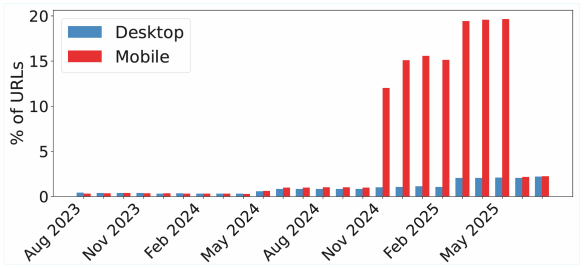


Figure 6: Chrome Platform Status [60] shows a surge in `RTCPeerConnection.SetLocalDescription` usage on popular websites, aligning with Meta Pixel's active WebRTC use between Nov.'24-May'25.

Table 2: Number of sites using localhost channels per crawl. *: Performed after our responsible disclosure. Red percentages report observed usage drop.

Crawl	Location	WebRTC	HTTP(S)	WebSocket
Android (April 2025)	USA	17,368	1,379	35
	EU	15,819	1,327	29
Desktop (May 2025)	USA	385	109	43
	EU	349	106	41
Android (June 2025∗)	USA	437 (-97,5%)	247 (-82,1%)	36
	EU	369 (-97,7%)	262 (-80.3%)	37
iOS (June 2025∗)	USA	498	220	30
	EU	422	250	29

velopers also remarked on Meta's switch from HTTP to WebSocket for localhost requests, and at the end of October/beginning of November, they noted that the issue appeared to be fixed. We hypothesize that these concerns ceased because WebRTC-based localhost data flows are visible only through Chrome's internal debugging interface (`chrome://webrtc-internals`) and not through Chrome's standard DevTools/Network tab, substantially reducing developer visibility. No further update was posted after Meta switched to WebRTC.

5.3 Crawl Results

Table 2 summarizes the results of various crawls (see §4.2). Notably, the initial (April 2025) crawl emulating an Android phone finds significantly more sites using localhost WebRTC and HTTP(S) channels than the Windows crawls, highlighting the Android-specific focus of Yandex's and Meta's campaigns. The iOS and the second Android crawls (June 2025) were performed after our responsible disclosure, when Meta and Yandex discontinued localhost-based tracking.

When visited from the EU and the USA, our crawls identify 15,819 and 17,368 sites establishing localhost connections, with Meta accounting for 99.1% of EU cases and 99.2% of the USA cases. Turning to HTTP(S), we observe HTTP-based localhost communication on 1,327 in the EU and 1,379 sites from the USA, of which Yandex is responsible for 95.0% and 95.1% of these cases, respectively. WebSocket-based localhost communication is rare, appearing only on 29 to 43 sites in our crawls, which were conducted after Meta switched from WebSockets to WebRTC methods. We discuss non-tracking related potential usage of localhost communications in §5.4.

These figures represent conservative lower bounds, as our crawling campaigns may be affected by bot detection and were limited to visiting only the landing page of each site in specific time periods. In practice, the scale of deployment is substantially larger according to BuiltWith [38], a website that tracks web technology adoption; it is estimated that the Meta Pixel and Yandex Metrica are present on over 5.8 million and 3M websites, respectively [39, 40].

User Consent. We conduct a complementary web crawl over the 16,831 (EU) and 18,431 (USA) sites where we observed localhost communications of any type in the initial (April 2025) Android crawls. The goal of this focused crawl was to assess whether localhost communications are triggered on page load or after getting user consent. We find that on 75.8% (EU), 78.2% (USA) of websites, localhost communications were triggered without user consent by Meta Pixel. Yandex Metrica triggered localhost communications on 84.4% (EU), 83.5% (USA) of websites without user consent. This raises concerns about whether users' privacy choices and data protection laws are respected (§9).

5.4 Other Uses of Localhost Communications

Our multi-modal web and app analysis methods identify other actors using localhost channels. However, their presence is marginal (below ten sites), hindering their ability to perform large-scale user tracking comparable to Meta and Yandex. To distinguish legitimate uses of localhost sockets (e.g., authentication) from tracking abuse, we reason about the organizations responsible and manually examine request paths, ports, parameters, and payloads to check for signs akin to the localhost abuse described in the threat model.

Local Web Development. Localhost is commonly used by developers for testing and debugging their websites. In a handful of cases, we find these debugging features to be left enabled in production.

OAuth. Some mobile apps perform user authentication in the browser, a recommended pattern in the OAuth specification [44]. After successful user authentication on the web, a token is passed back to the native app via a defined port on `127.0.0.1`. Crawl results show that scripts from Okta [106], an identity provider, initiate connections to *authenticator-localprod.com*, which resolves to `127.0.0.1`.

Anti-Fraud. Anti-fraud solutions often scan localhost ports to detect whether the host is controlled remotely. In the past, eBay was found scanning visitors' open ports [2]. We identify a script from SEON (seondf.com) [118], an anti-fraud firm, actively probing 13 different ports using WebRTC.

Video Streaming. Zalando (de.zalando.mobile) and Taobao (com.taobao.taobao) apps spin up a temporary localhost HTTP server to route video streams through it. The localhost server fetches the content from a real CDN resource before sending it back to a WebView-based player via localhost.

Other Cases. Several subdomains from the cross-platform tracking firm 33Across [12] like *cdn-ima.33across.com* resolve to `127.0.0.1` in Europe, but not in the USA. We hypothesize that the organization uses DNS-based methods to block access from different regions.

6 Affected Browsers and Mitigations

We reproduce HTTP and WebRTC-based web-to-mobile ID sharing methods to test whether popular browsers—i.e., Chrome, Edge, Mozilla Firefox, DuckDuckGo and Brave for Android—are vulnerable to Yandex and Meta's web-to-mobile bridging. As Table 3 shows, all tested browsers were at least partially affected. Below, we discuss several browser-level mitigations to limit localhost-based tracking and abuse, some of which were introduced in response to our disclosure.

Blocklists. Brave and DuckDuckGo browsers implement blocklists to block resources from known trackers [37, 45].

Table 3: Overview of browsers affected by the localhost tracking method. $*$ Uses blocklists to block Yandex and Meta scripts/domains.

Browser	Version	HTTP(S)	WebRTC
Chrome	136.0.7103.125	Affected	Affected
Edge	136.0.3240.50	Affected	Affected
Firefox	138.0.2	Affected	Partial
DuckDuckGo	5.233.0	Affected$*$	Affected$*$
Brave	1.78.102	Partial$*$	Affected$*$

In our measurements, Brave blocks all domains used by both Yandex and Meta, while DuckDuckGo misses three infrequently used, alternate Yandex domains. DuckDuckGo updated its blocklist after our disclosure. We stress that domain blocklisting does not restrict the underlying communication channels and may leave users vulnerable to potential localhost abuse due to incomplete blocklists.

Blocking Localhost Requests. Brave blocks HTTP(S) requests to `127.0.0.1` and `localhost` [36], hence preventing the HTTP request method used by Yandex Metrica and the early iteration of Meta Pixel. However, Brave does not block requests to domains such as yandexmetrica.com that resolve to `127.0.0.1` [4] or localhost WebRTC requests.

Firefox & Enhanced Tracking Protection (ETP). Firefox ETP provides privacy protections beyond blocklists, including anti-fingerprinting [102]. Our measurements show that ETP blocks the Meta Pixel only in Strict mode. However, Meta's STUN-based localhost method fails on Firefox even when the script loads, without producing an explicit error message. Although the TURN-based WebRTC method later introduced by Meta would likely have succeeded, Meta terminated the campaign before a complete rollout. Similar to DuckDuckGo Browser, Firefox misses three alternate Yandex domains, even when the strict ETP mode is enabled.

Disallowing Mixed Content From Localhost. While browsers generally block active mixed content [87], they do not necessarily block it when coming from the localhost, considered a "potentially trusted" context [91, 128]. While Safari prevents active content such as HTML from being loaded via localhost, our experiments show that it allows web-to-app channels for loading passive content such as images. We also find that requests to domains resolving to the localhost (e.g., *yandexmetrica.com*) over HTTPS are not blocked in Safari.

Restricted Ports. Chrome maintains a list of restricted ports, including TCP:22 (SSH) and 25 (SMTP) [62]. As a short-term mitigation, Chrome added the localhost ports used by Meta and Yandex to the list of restricted ports through a Finch (public) trial [43, 59, 61]. We verify the effectiveness of this mitigation against Meta and Yandex scripts, confirming the blocked connections in stderr output and console logs.

Local Network Access (LNA). First released in Chrome stable v142 (October 28th, 2025), Local Network Access (LNA) standard enables users to allow or reject access the local network or loopback addresses [122]. LNA defines two permissions, `local-network` and `loopback-network`, replacing the previous single permission covering both cases. LNA does not automatically block the underlying localhost channels, but it gates the access behind a user permission. This delegation of responsibility raised usability concerns, particularly due to prompt text being hard to grasp by regular users [6].

LNA handles HTTP(S) requests, even if the requests are sent to a domain resolving to `127.0.0.1`. Other protocols including WebSockets, WebRTC and WebTransport are experimentally supported with a command line flag. While the LNA editors consider various protocols that can be used for localhost and local network access, we discuss potential blind spots in the following section. As of January 2026, LNA has been only shipped in Chrome Stable (version 142) and Firefox Nightly (version 143) [11]. WebKit (Safari) has also expressed their intent to ship LNA [7].

7 Additional Attack Vectors

After observing browsers' short-term mitigations following our public disclosure of the Meta and Yandex findings, we investigate alternative techniques capable of bypassing these new defenses. While these additional attack vectors remain technically exploitable, we did not observe evidence of abuse via web crawls. We responsibly disclosed the new vulnerabilities to the relevant browser makers prior to paper submission.

7.1 WebRTC & IPv6 Addresses

LNA determines whether to block a connection by checking if the destination IP corresponds to a private IPv4/v6 address or the loopback address. We develop a variant of Meta's WebRTC STUN method that bypasses Chrome's LNA implementation [63] by leveraging the local ICE candidates generated when invoking *setLocalDescription*. For comparison, Meta's method hardcoded the loopback IP address in its SDP answer to trigger a localhost connection.

We force our WebRTC peer to select one of its own generated ICE candidate addresses alongside a chosen port by setting it as the answer received from a second peer on the same page, effectively sending a WebRTC Binding Request to localhost at the chosen port. A native app can bind to the `0.0.0.0` or `::1` address and listen for these messages on the chosen port. Our test also reveals that the generated candidates consist of a private IPv4 address and a global unicast IPv6 address, if supported. To evade LNA, which mediates traffic to private addresses, an attacker can use the global unicast IPv6 address in the list of generated candidates. On the mobile versions of the Chrome (v138.0.7204.180) and Firefox (v141.0.2) browsers, scripts can directly read the IP addresses from ICE candidates and thus choose the IPv6 address. On desktop browsers and the mobile version of Brave, the attacker cannot directly check if the given ICE candidate address is a private address or not due to the introduction of random mDNS (.local) addresses to prevent private IP address leaks via WebRTC [10]. However, our tests show that the private addresses appear first in the generated candidate list, and the global unicast IPv6 address is placed second on both Chrome and Brave. Thus, if two or more candidates are generated, scripts could trivially select the last `UUID.local` address. Similar concerns about the use of IPv6 Global Unicast addresses to bypass LNA have been raised independently [105].

7.2 mDNS Lookups as a Web-to-App Channel

As discussed in §2.3, mDNS queries are resolved via multicast to `224.0.0.251:5353` for IPv4. As a result, any apps listening to this address can receive queries coming from other processes running on the same device, enabling an additional unvetted channel between browser and native contexts. We identify three additional mDNS-based web-to-app ID sharing methods that go beyond the known shortcomings of the LNA [119]:

- The first one exploits this behavior by issuing a fetch request to an attacker-chosen `.local` hostname that encodes sensitive data. An app listening for mDNS queries can view the address and extract the data from it.
- A second variant abuses WebRTC-generated `UUID.local` hostnames and server-side bridging, which are intended to prevent private IP address discovery [127]. In this case, a web script gathers ICE candidates for a WebRTC peer, triggering an mDNS multicast query. It then reads the gathered `UUID.local` addresses and sends these to a server. The native app also receives and sends these addresses to the server. As `UUID.local` addresses are unique, they effectively enable the linkage of browser visits to device IDs obtained by apps.
- In the third variant, the script creates a custom .local address and adds it to WebRTC ICE candidates. This process will trigger another mDNS multicast query which can be received by the app.

8 Discussion

Mitigating privacy threats like localhost abuse requires addressing structural changes across browsers, mobile platforms, and standards, including the definition of new threat models. Our study shows that localhost connections can be repurposed as a covert channel to bridge web and native mobile app contexts. The abuse violates fundamental trust assumptions and isolation mechanisms of current mobile platforms and web browsers. In response to our disclosures, multiple browser engineers confirmed that localhost-based web-mobile bridging was not explicitly considered in their threat models, highlighting structural shortcomings and a blind spot. The impact of this abuse is significantly amplified when a single actor with substantial web and mobile reach can deploy coordinated tracking logic across web analytics scripts, mobile SDKs and widely installed mobile applications. In the cases of the Meta Pixel and Yandex Metrica, their significant market presence enabled silent, large-scale linking of web activity to user identities or unique device IDs.

Recent platform responses reflect growing awareness, but also reveal important structural limitations. Historically, both LAN and localhost access were implicitly covered by install-time network access permissions [57]. In early 2025, Android 16 (beta) introduced an experimental opt-in Local Network Access (LNA) protection, gating LAN socket access behind `NEARBY_WIFI_DEVICES`, with a dedicated runtime permission planned under `NEARBY_DEVICES` [21]. iOS applies comparable gating and, since iOS 14, prompts users for Local Network access [29].

Current mobile platform permissions do not restrict explicit access to localhost sockets, while the browser-side Local Network Access (LNA) specification attempts to regulate its access as part of its local-network threat model. We believe that platform defenses must clearly decouple localhost socket access from LAN access, both in permission semantics and usability considerations, despite potential interference with legitimate use cases [6]. Conflating the scope and purposes of these IP ranges risks consent fatigue and user confusion, while leaving localhost-based channels insufficiently constrained. Beyond OS-level controls, standardization bodies such as the W3C and the IETF should revisit the trust assumptions surrounding localhost, particularly in specifications that implicitly grant it elevated status without accounting for adversarial cross-process use.

Our results suggest that platform policies and enforcement mechanisms play a critical role in deterring abuse. While Android is more flexible with respect to background execution, iOS introduces stricter constraints on long-lived background execution, reducing the feasibility of persistent localhost listeners and limiting them only to specific purposes (e.g., audio playback or navigation).

The hybrid and coordinated nature of new cross-platform privacy abuses, and the lack of appropriate threat models complicate detection and enforcement at scale. Large-scale detection of coordinated cross-context abuses could be limited by: (i) the combinatorial explosion inherent to testing millions of potential websites and apps in combination; (ii) the use of evasion techniques; and (iii) the need to jointly exercise apps and websites to trigger coordinated behaviors. As we demonstrated, combining static and dynamic analysis methods across both web and mobile contexts can systematically reduce the problem space and identify candidate app-website sets exhibiting signals of coordinated cross-context behavior that warrant deeper inspection.

9 Legal Considerations

What does the law in the EU say about the behavior by Meta and Yandex? Because of length constraints, we discuss only EU law, and within the EU we focus on the main legal principles, omitting greater detail.

A main requirement of the GDPR is that companies offer transparency about what they do with personal data (article 5(1)(a)) [52]. We did not find any mention of the localhost communication method and its use in Meta or Yandex' websites, privacy policies, or other public documents. Hence, transparency is lacking.

The GDPR also requires a company to have a 'legal basis' for personal data processing. The GDPR contains six possible legal bases; the 'legitimate interests provision' and the data subject's 'consent' are most relevant for this paper (GDPR article 6). In general, a company can rely on the 'legitimate interests provision' if the company uses personal data for its legitimate interests and those interests are not overridden by the data subject's privacy or other interests (article 6(1)(f)). In this case, the companies cannot rely on this provision, among other reasons because the data subject's interests outweigh the company's interests, because of the lack of transparency, and because the companies did not offer a clear opt-out possibility.

Another possible legal basis is the data subject's consent (article 6(1)(a)). The requirements for valid consent are strict. For example, consent must be 'informed' and 'specific', and the data subject must clearly indicate their agreement (article 4(11)). In this case, the lack of transparency about the tracking method makes it impossible for the companies to rely on consent, because the data subject's consent cannot be informed or specific. Hence, the companies do not have a legal basis for the personal data processing, which implies a breach of the GDPR.

The companies' tracking behavior probably also violates the ePrivacy Directive [51], which requires that any party who stores information on a user device, or gains access to information already stored on a user device, ask the user for informed consent first. The ePrivacy Directive applies to the storing and accessing of cookies, but also to other information, such as (web) pixels [50].

Meta might have also violated the Digital Services Act (article 26(3) and 28(2)) [54] and the Digital Markets Act (article 5(2)) [53], but a discussion of those laws falls outside the scope of this analysis.

10 Related Work

Local Network Access. Kuchhal and Li observed widespread localhost and LAN access in the wild, driven by both intentional uses (e.g., fraud detection) and accidental leaks, with behavior varying by OS [85]. Their crawlers were based on desktop browsers, which may explain why they did not encounter the mobile-focused localhost tracking methods. Beyond browsers, multiple studies show that smart-home devices and mobile apps use local discovery protocols like mDNS, SSDP, and UPnP to discover nearby devices or services [34, 57, 116, 117]. Public incidents further stress how browsers can reach privileged localhost services: Zoom's macOS client exposed a local server enabling RCE via crafted URLs [32]. Researchers discovered eBay scanning localhost ports via WebRTC [2]. Oligo Security disclosed a critical RCE in Anthropic's MCP Inspector via unauthenticated localhost access [109]. ASUS DriverHub exposed a service that allowed remote code execution from malicious websites [120]. All these cases establish that browser-originated traffic to local endpoints is an exploitable vector. Our work extends these findings by uncovering a new privacy threat that bridges web and native tracking paradigms.

Tracking on Mobile and Web Platforms. Extensive research has shown that both mobile and web ecosystems leak identifiers due to opaque third-party code, limited transparency, and gaps in policy enforcement [13, 15, 31, 49, 82–84, 104, 112]. However, these efforts analyzed abuse following a platform-centric approach, thus missing coordinated privacy abuses like localhost-based tracking. On mobile, several studies have examined how embedded SDKs exploit local interfaces to exfiltrate data. Jia *et al.* systematically analyzed 24K Android apps and uncovered widespread open-port usage with exploitable configurations [80], while Wu *et al.* found that roughly 15% of popular apps — including Yandex apps — expose TCP services, often through embedded SDKs [132]. In parallel, the web ecosystem continues to rely on pervasive tracking techniques such as evercookies and fingerprinting [13, 49]. Additionally, WebRTC features have been exploited for device and network enumeration, IP leakage, and local fingerprinting, enabling cross-site correlation attacks [1, 49]. Within this landscape, Meta's tracking infrastructure — particularly the Pixel and Conversions API — has been under scrutiny for bypassing browser-level defenses and enabling persistent user tracking [35, 48]. Recently, Weerasekara *et al.* showed that Android WebViews expose JavaScript-native bridges abused to synchronize native IDs and session state with embedded web content, including Yandex' and Meta's WebViews [131]. Beer *et al.* showed that Android CustomTabs weaken origin isolation and facilitate ID flows via shared storage, referrers, or injected interfaces [33].

11 Conclusions

This paper revealed a new threat affecting web browsers and mobile platforms: the widespread use of a previously undisclosed form of tracking paradigm to bridge web and native mobile identities, nullifying protections such as app sandboxing, browsers' Incognito Mode, and cookie clears. Our longitudinal analysis showed that this tracking vector has remained unnoticed for eight years in the case of Yandex. Two Meta Android apps with 15 billion combined installs also remained undetected for eight months until our disclosures. This shows that studying web and mobile privacy threats in isolation as disjoint research problems leads to structural blind spots across web and mobile platforms, from their threat models to app verification processes. By evaluating existing and upcoming defenses, we demonstrated additional web-to-app ID sharing vectors such as mDNS lookups, that remain exploitable but are not abused in the wild. Our study led to mitigations deployed by major browser vendors, and the termination of tracking campaigns by Yandex and Meta. We encourage researchers, regulators, and relevant software and platform vendors to focus on similar coordinated cross-context privacy risks that may have remained under the radar.

Acknowledgments

We thank Chrome, Mozilla, DuckDuckGo, and Brave for their collaboration. Special thanks to Álvaro Feal for helping us reproduce app behaviors, Tom Van Goethem, Martin Thomson, Steve Englehardt, and Dave Harbage for diligently handling our disclosure and providing valuable insights, Bart Preneel for helping with media outreach, HTTP Archive Project for their longitudinal dataset, and Schloss Dagstuhl – Leibniz Center for Informatics for sparking this collaboration. This research was partially supported by the MICIU/AEI/10.13039/501100011033/ through the grants PID2022-143304OB- I00, EU ERDF (PARASITE), the Flemish Government through the Cybersecurity Research Program grant VOEWICS02, the EU Horizon 2020 programme under grant agreement H2020-SC1-FA-DTS-2018-1-826284 (ProTego), the Research Council KU Leuven IF/C1 "From Website Fingerprinting to App Fingerprinting: Inferring private user activity from encrypted network traffic", the Austrian Funding agency FFG under grant S3AI Ref. 872172 (COMET Module) and by the Defense Advanced Research Projects (DARPA), contract number FA8750-19-C-0502 (RACE PRISM). Gunes Acar is supported by a Netherlands Organisation for Scientific Research (NWO) Vidi grant. Prof. N. Vallina-Rodriguez was appointed as the 2019 Ramon y Cajal fellow (RYC2020-030316-I), funded by the MICI-U/AEI/10.13039/501100011033/ and the EU ESF Investing in your Future.

Ethical Considerations

Our research identified an unreported cross-context tracking technique that undermines expected sandboxing and isolation guarantees in mobile operating systems and browsers, and poses a systemic privacy risk to users. To that end, our ethical objective was ecosystem-wide risk reduction through disclosures to relevant stakeholders. A timeline of our disclosures can be found in Table 4.

Stakeholders and Risk Framing. We identify four stakeholder groups: (i) users whose browsing activity can be linked to native-app identities, (ii) browser and OS vendors — both Android and iOS — who define and enforce isolation boundaries and other privacy controls, (iii) website operators who embed third-party scripts and may face operational fallout from mitigations, and (iv) companies whose scripts and apps implement localhost tracking (Meta and Yandex). We treated browser and OS vendors as the primary parties for disclosure (rather than Meta and Yandex), because the issue reflects an intentional abuse of cross-context isolation security mechanisms rather than a product defect. The risk is systemic: even if one actor discontinues the practice, the same pattern can recur unless platforms constrain misuse of localhost access.

Responsible Disclosure Process and Timeline. We first disclosed the localhost abuse to the Android Security and Chrome Privacy teams following responsible disclosure practices, as the issue was initially observed on their products. After confirming the behavior across other browsers, we extended disclosure to Mozilla, Brave, DuckDuckGo, Microsoft Edge, WebKit, and to EU- and US-based data protection authorities. Each disclosure included detailed technical reports, proof-of-concept applications, screen captures, and analysis scripts to support independent validation and reproduction.

Our disclosures were an iterative process rather than a single event. We maintained active communication with parties, and responded to technical questions, conducted additional experiments, discussed and tested proposed mitigations, and provided updates as new vectors were identified. In several cases, vendors independently replicated our findings.

Public Disclosure. Because the tracking technique was actively abused, we pursued public disclosure in parallel with affected vendor coordination before submission of this paper. Public disclosures increase societal awareness, and motivate and expedite mitigations. The public disclosure occurred through a dedicated website (https://localmess.github.io) with independent journalistic reporting. As part of the same release process, reporters asked Meta and Yandex's statements the day before the publication, giving them a chance to respond. Overall, we believe that this disclosure process primarily benefited web and mobile users, consistent with the Menlo Principles of Beneficence and Public Interest, also fostering regulatory and legislative debate. The termination of the observed campaigns and the deployment of mitigations reduced future privacy and security risks for billions of users. The findings have since informed public accountability mechanisms, including class action lawsuits in multiple jurisdictions, expert testimony before the Spanish Congress in the context of the Digital Service Act (DSA), and technical discussions at IETF and W3C, where LNA standardization documents cite this work [8] While we omitted the mention of our public disclosure in our initial paper submission to preserve anonymity and unbiased reviews, we informed USENIX'26 Program Chairs about the public status of our findings.

Table 4: Disclosure timeline.

Date	Action
End of Jan. 2025	Initial discovery of web-to-app identifier bridging via localhost.
Mar. 2025	Private disclosure to Android Security and Chrome Privacy teams.
Apr.–May 2025	Disclosure to additional browser vendors (Mozilla, Brave, DuckDuckGo, Edge) and to representatives of EU and USA Data Protection Authorities.
May 2025	Discovery and disclosure of additional bypass vectors (e.g., TURN-based variants) to affected vendors.
End May 2025	Rollout of first (temporary) browser mitigations.
Jun. 2025	Coordinated public disclosure; Meta and Yandex terminate the observed behavior.
	Disclosure of a localhost-based cross-profile data leakage vector between Android Work and Personal profiles.
Aug. 2025	Disclosure of the additional localhost vectors.
Aug.-Oct. 2025	Deployment of Local Network Access (LNA) restrictions across major browsers.

Disclosure of Additional Vectors During Standardization. We publicly disclosed newly identified LNA bypass techniques (§7) via WICG issue trackers [5,9] rather than private disclosures, because LNA was still under active standardization and its editors explicitly encouraged public input. We did not observe these vectors being exploited in the wild. Open disclosures allowed mitigations to be designed and deployed before exploitation, without introducing fundamentally new capabilities, since similar vectors were already discussed. Early disclosures also enabled abuse monitoring by vendors and input by the security community.

Data Handling and Harm Minimization. Our measurements were designed to minimize potential harm. Crawls avoided concurrent visits to the same site to reduce server load and lower the chance of operational disruption. We did not use authenticated user accounts and limited interactions to what was necessary to observe default script behavior.

Potential Negative Outcomes (A Retrospective Look). While our study and related disclosures led to several positive outcomes, such as the termination of the active abuse campaigns and the development of mitigations, it may also have potential negative secondary effects. For instance, as is typical with large security updates, restrictions due to LNA caused breakage on certain sites [3]. Yet, we consider that the privacy harm affecting billions of users outweighed it. Public disclosures of LNA bypasses could lead to abuse, but as mentioned above, similar vectors were already openly discussed and early disclosure supports mitigation design and abuse monitoring.

Open Science

Data and code from our study can be found on https://doi.org/10.5281/zenodo.17880051. Because the involved companies discontinued the disclosed tracking practices, exact replication of abuse behavior remains possible through static analysis of publicly archived APKs and HAR files, as well as our PoC websites and apps that reproduce the localhost-probing behavior. To that end, we share the crawler code, crawl data, crawl analysis scripts, Frida scripts, and proof-of-concept apps and webpages.

References

[1] STUN IP Address Requests for WebRTC. https://github.com/diafygi/webrtc-ips, 2015.

[2] eBay is port scanning visitors to their website - and they aren't the only ones - nem.ec. https://blog.nem.ec/2020/05/24/ebay-port-scanning, 2020.

[3] Blink Security Feature: Local Network Access issues. https://issues.chromium.org/issues?q=customfield1222907:%22Blink%3ESecurityFeature%3ELocalNetworkAccess%22, 2025.

[4] Blocking LAN access from non-LAN sites doesn't seem to be working correctly. https://github.com/brave/brave-browser/issues/46573, 2025.

[5] Consider some reserved IPv6 ranges as local. https://github.com/WICG/local-network-access/issues/15#issuecomment-3225316474, 2025.

[6] LNA issues with split DNS and unmanaged clients. https://issues.chromium.org/issues/457794553, 2025.

[7] Local Network Access · Issue #163 · WebKit/standards-positions. https://github.com/WebKit/standards-positions/issues/163, 2025.

[8] Local Network Access – Draft Community Group Report. https://wicg.github.io/local-network-access/#user-mediation, 2025.

[9] mDNS access for WebRTC. https://github.com/WICG/local-network-access/issues/22#issuecomment-3223162671, 2025.

[10] PSA: Private IP addresses exposed by WebRTC changing to mDNS hostnames. https://groups.google.com/g/discuss-webrtc/c/6stQXi72BEU, 2025.

[11] Request for Mozilla Position on an Emerging Web Specification: Local Network Access. https://github.com/mozilla/standards-positions/issues/1260, 2025.

[12] 33Across. Home. https://www.33across.com, 2025.

[13] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proc. of ACM CCS*, 2014.

[14] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based attacks to discover and control local IoT devices. In *Proc. of IoT S&P Workshop*. ACM, 2018.

[15] Noura Alomar, Joel Reardon, Aniketh Girish, Narseo Vallina-Rodriguez, and Serge Egelman. The Effect of Platform Policies on App Privacy Compliance: A Study of Child-Directed Apps. *Proc. on PETS*, (3), 2025.

[16] Harald Alvestrand. PSA: Modification of SDP "ufrag" and "passwd" attributes is going away. https://groups.google.com/g/discuss-webrtc/c/PIJZN5MTZF4, 2025.

[17] Android. Android Work Profiles. https://www.android.com/intl/en_au/enterprise/work-profile/, 2025.

[18] Android Developers. Features and APIs Overview — Android 12. https://developer.android.com/about/versions/12/features, 2025.

[19] Android Developers. Foreground Service Types. https://developer.android.com/develop/background-work/services/fgs/service-types, 2025.

[20] Android Developers. Foreground Services Overview. https://developer.android.com/develop/background-work/services/fgs, 2025.

[21] Android Developers. Local Network Permission (Android Privacy and Security). https://developer.android.com/privacy-and-security/local-network-permission, 2025.

[22] Android Developers. Permissions on Android (Overview). https://developer.android.com/guide/topics/permissions/overview, 2025.

[23] Android Developers. Processes and Threads. https://developer.android.com/guide/components/processes-and-threads, 2025.

[24] Android Open Source Project (AOSP). Application Sandbox. https://source.android.com/docs/security/app-sandbox, 2025.

[25] Apple Developer Documentation. App Tracking Transparency. https://developer.apple.com/documentation/apptrackingtransparency, 2025.

[26] Apple Developer Documentation. Configuring App Groups. https://developer.apple.com/documentation/xcode/configuring-app-groups, 2025.

[27] Apple Developer Documentation. Configuring Background Execution Modes. https://developer.apple.com/documentation/xcode/configuring-background-execution-modes, 2025.

[28] Apple Developer Documentation. Entitlements (Bundle Resources). https://developer.apple.com/documentation/bundleresources/entitlements, 2025.

[29] Apple Developer Documentation. NSLocalNetworkUsageDescription (Bundle Resources — Information Property List). https://developer.apple.com/documentation/bundleresources/information-property-list/nslocalnetworkusagedescription, 2025.

[30] Apple Developer Documentation. Supporting Associated Domains. https://developer.apple.com/documentation/xcode/supporting-associated-domains, 2025.

[31] Ioannis Arkalakis, Michalis Diamantaris, Serafeim Moustakas, Sotiris Ioannidis, Jason Polakis, and Panagiotis Ilia. Abandon All Hope Ye Who Enter Here: A Dynamic, Longitudinal Investigation of Android's Data Safety Section. In *Proc. of USENIX Security*, 2024.

[32] Assetnote. Zoom Zero-Day Followup: Getting the RCE. https://www.assetnote.io/resources/research/zoom-zero-day-followup-getting-the-rce, 2019.

[33] Philipp Beer, Marco Squarcina, Lorenzo Veronese, and Martina Lindorfer. Tabbed Out: Subverting the Android Custom Tab Security Model. In *Proc. of IEEE S&P Symposium*, 2024.

[34] Angelos Beitis, Jeroen Robben, Alexander Matern, Zhen Lei, Yijia Li, Nian Xue, Yongle Chen, Vik Vanderlinden, and Mathy Vanhoef. LANShield: Analysing and Protecting Local Network Access on Mobile Devices. *Proc. on PETS*, 2025.

[35] Paschalis Bekos, Panagiotis Papadopoulos, Evangelos P. Markatos, and Nicolas Kourtellis. The Hitchhiker's Guide to Facebook Web Tracking with Invisible Pixels and Click IDs. In *Proc. of WWW*, 2023.

[36] Brave Software. Brave Privacy Update: Localhost Resource Permission. https://brave.com/privacy-updates/27-localhost-permission, 2023.

[37] Brave Software. Brave Shields. https://brave.com/shields, 2025.

[38] BuiltWith Trends. BuiltWith. https://builtwith.com, 2025.

[39] BuiltWith Trends. Websites using Facebook Pixel. https://trends.builtwith.com/websitelist/Facebook-Pixel, 2025.

[40] BuiltWith Trends. Websites using Yandex Metrika. https://trends.builtwith.com/analytics/Yandex-Metrika, 2025.

[41] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. Technical Report RFC 6763, IETF, 2013.

[42] Stuart Cheshire and Marc Krochmal. Multicast DNS. Technical Report RFC 6762, IETF, 2013.

[43] Chrome for Developers. What is a Chrome Finch experiment? https://developer.chrome.com/docs/web-platform/chrome-finch, 2025.

[44] William Denniss and John Bradley. OAuth 2.0 for Native Apps, Section 4.1. https://datatracker.ietf.org/doc/html/rfc8252#section-4.1, 2017.

[45] DuckDuckGo. Tracker Blocklists. https://github.com/duckduckgo/tracker-blocklists, 2022.

[46] DuckDuckGo. Tracker Radar Collector. https://github.com/duckduckgo/tracker-radar-collector, 2025.

[47] Zakir Durumeric. crux-top-lists. https://github.com/zakird/crux-top-lists, 2022.

[48] Asmaa El Fraihi, Nardjes Amieur, Walter Rudametkin, and Oana Goga. Client-side and Server-side Tracking on Meta: Effectiveness and Accuracy. *Proc. on PETS*, (3), 2024.

[49] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proc. of ACM CCS*, 2016.

[50] European Data Protection Board. Guidelines 2/2023 on Technical Scope of Art. 5(3) of ePrivacy Directive (Version 2). https://www.edpb.europa.eu/system/files/2024-10/edpb_guidelines_202302_technical_scope_art_53_eprivacydirective_v2_en_0.pdf, 2024.

[51] European Parliament and Council. Directive 2002/58/EC (ePrivacy Directive), as amended by Directive 2009/136/EC. https://eur-lex.europa.

eu/LexUriServ/LexUriServ.do?uri=CONSLEG:2002L0058:20091219:EN:HTML, 2009.

[52] European Parliament and Council. General Data Protection Regulation, Regulation 2016/679. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679, 2016.

[53] European Parliament and Council. Regulation (EU) 2022/1925 on Contestable and Fair Markets in the Digital Sector (Digital Markets Act). http://data.europa.eu/eli/reg/2022/1925/2022-10-12, 2022.

[54] European Parliament and Council. Regulation (EU) 2022/2065 on a Single Market for Digital Services (Digital Services Act). http://data.europa.eu/eli/reg/2022/2065/oj, 2022.

[55] Álvaro Feal, Julien Gamba, Juan Tapiador, Primal Wijesekera, Joel Reardon, Serge Egelman, and Narseo Vallina-Rodriguez. Don't accept candy from strangers: An analysis of third-party mobile sdks. In *Data Protection and Privacy: Data Protection and Artificial Intelligence*, 2021.

[56] Julien Gamba, Álvaro Feal, Eduardo Blazquez, Vinuri Bandara, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. Mules and permission laundering in Android: Dissecting custom permissions in the wild. *IEEE Transactions on Dependable and Secure Computing*, 2023.

[57] Aniketh Girish, Tianrui Hu, Vijay Prakash, Daniel J. Dubois, Srdjan Matic, Danny Yuxing Huang, Serge Egelman, Joel Reardon, Juan Tapiador, David Choffnes, and Narseo Vallina-Rodriguez. In the Room Where It Happens: Characterizing Local Communication and Threats in Smart Homes. In *Proc. of ACM IMC*, 2023.

[58] Aniketh Girish, Joel Reardon, Juan Tapiador, Srdjan Matic, and Narseo Vallina-Rodriguez. Your Signal, Their Data: An Empirical Privacy Analysis of Wireless-scanning SDKs in Android. *Proc. on PETS*, 2025.

[59] Google Chrome Developers. Add support for UDP sockets for localhost restrictions (6574417) · Gerrit Code Review. https://chromium-review.googlesource.com/c/chromium/src/+/6574417, 2025.

[60] Google Chrome Developers. Chrome Platform Status: HTML & JavaScript usage metrics. https://chromestatus.com/metrics/feature/timeline/popularity/3452, 2025.

[61] Google Chrome Developers. Enable RestrictAbusePortsOnLocalhost by default (6607695) · Gerrit Code Review. https://chromium-review.googlesource.com/c/chromium/src/+/6607695, 2025.

[62] Google Chrome Developers. net/base/port_util.cc - chromium/src.git - Git at Google. https://chromium.googlesource.com/chromium/src.git/+/refs/heads/master/net/base/port_util.cc#30, 2025.

[63] Google Chrome Developers. New Permission Prompt for Local Network Access. https://developer.chrome.com/blog/local-network-access, 2025.

[64] Google Chrome Developers. Overview of CrUX. https://developer.chrome.com/docs/crux, 2025.

[65] Google Play. Developer Program Policy. https://support.google.com/googleplay/android-developer/answer/9857753, 2023.

[66] Google Play. User Data Policy for Android Developers. https://support.google.com/googleplay/android-developer/answer/10144311?sjid=18338717510598425318-EU, 2024.

[67] Google Play Console Help. Advertising ID. https://support.google.com/googleplay/android-developer/answer/6048248, 2025.

[68] Google Play Console Help. Advertising ID. https://support.google.com/googleplay/android-developer/answer/6048248, 2025.

[69] Google Play Console Help. Data safety on Google Play. https://support.google.com/googleplay/android-developer/answer/9857753, 2025.

[70] Google Play Console Help. Provide information for Google Play's Data safety section. https://support.google.com/googleplay/android-developer/answer/10787469?hl=en, 2025.

[71] Google Policy Center. Device and Network Abuse. https://support.google.com/googleplay/android-developer/answer/16273414, 2025.

[72] Google Project Zero. Issue 42452214. https://project-zero.issues.chromium.org/issues/42452214, 2016.

[73] Philipp Hancke. Not a Guide to SDP Munging. https://webrtchacks.com/not-a-guide-to-sdp-munging/, 2020.

[74] HTTP Archive. Cookies. https://almanac.httparchive.org/en/2024/cookies#top-first-and-third-party-cookies-and-domains-setting-them, 2024.

[75] HTTP Archive. The HTTP Archive. https://httparchive.org, 2025.

[76] Internet Engineering Task Force. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. https://datatracker.ietf.org/doc/html/rfc8445, 2018.

[77] Internet Engineering Task Force. JavaScript Session Establishment Protocol (JSEP). https://datatracker.ietf.org/doc/html/rfc8829#name-modifying-an-offer-or-answe, 2021.

[78] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from DNS rebinding attacks. *ACM Transactions on the Web (TWEB)*, 3(1):1–26, 2009.

[79] Nikhil Jha, Martino Trevisan, Luca Vassio, and Marco Mellia. The Internet with Privacy Policies: Measuring The Web Upon Consent. In *ACM Trans. Web*, volume 16, 3. ACM, 2022.

[80] Yunhan Jack Jia, Qi Alfred Chen, Yikai Lin, Chao Kong, and Z. Morley Mao. Open Doors for Bob and Mallory: Open Port Usage in Android Apps and Security Implications. In *Proc. of IEEE EuroS&P*, 2017.

[81] Robin Kirchner, Jonas Möller, Marius Musch, David Klein, Konrad Rieck, and Martin Johns. Dancer in the dark: Synthesizing and evaluating polyglots for blind Cross-Site scripting. In *Proc. of USENIX Security*, 2024.

[82] Simon Koch, Benjamin Altpeter, and Martin Johns. The {OK} is not enough: A large scale study of consent dialogs in smartphone applications. In *Proc. of USENIX Security*, 2023.

[83] Konrad Kollnig, Pierre Dewitte, Max Van Kleek, Ge Wang, Daniel Omeiza, Helena Webb, and Nigel Shadbolt. A fait accompli? An empirical study into the absence of consent to third-party tracking in Android apps. In *Proc. of SOUPS*, 2021.

[84] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. Are iPhones Really Better for Privacy? Comparative study of iOS and Android Apps. In *Proc. on PETS*, 2022.

[85] Dhruv Kuchhal and Frank Li. Knock and talk: investigating local network communications on websites. In *Proc. of ACM IMC*, 2021.

[86] René Mayrhofer, Jeffrey Vander Stoep, Chad Brubaker, and Nick Kralevich. The Android platform security model. *ACM TOPS*, 24(3), 2021.

[87] MDN Web Docs. Mixed content. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content, 2025.

[88] MDN Web Docs. Permissions API. https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API, 2025.

[89] MDN Web Docs. RTCPeerConnection. https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection, 2025.

[90] MDN Web Docs. Same-Origin Policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy, 2025.

[91] MDN Web Docs. Secure contexts. https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts, 2025.

[92] MDN Web Docs. State Partitioning - Privacy on the web. https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/State_Partitioning, 2025.

[93] MDN Web Docs. Third-party cookies - Privacy on the web. https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Third-party_cookies, 2025.

[94] Meta for Developers. About advanced matching for web. https://www.facebook.com/business/help/611774685654668, 2025.

[95] Meta for Developers. Facebook SDK making call to localhost. https://web.archive.org/web/20250531105747/https://developers.facebook.com/community/threads/317050484803752/, 2025.

[96] Meta for Developers. fbp and fbc Parameters. https://web.archive.org/web/20250602140111/https://developers.facebook.com/docs/marketing-api/conversions-api/parameters/fbp-and-fbc/, 2025.

[97] Meta for Developers. Meta Pixel. https://developers.facebook.com/docs/meta-pixel, 2025.

[98] Meta for Developers. Meta Pixel in Android WebView. https://web.archive.org/web/20250531105711/https://developers.facebook.com/community/threads/937149104821259/, 2025.

[99] Meta Platforms, Inc. Cookies Policy. https://web.archive.org/web/20250602140832/https://www.facebook.com/privacy/policies/cookies/?subpage=subpage-1.3, 2025.

[100] Miki. The Power of DNS Rebinding: Stealing WiFi Passwords with a Website. https://blog.miki.it/posts/the-power-of-dns-rebinding-stealing-wifi-passwords-with-a-website/, 2015.

[101] Mozilla. Mitigate CSRF attacks against internal networks (block rfc 1918 local addresses from non-local addresses). https://bugzilla.mozilla.org/show_bug.cgi?id=354493, 2006.

[102] Mozilla Support. Enhanced Tracking Protection in Firefox for desktop. https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop, 2025.

[103] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share First, Ask Later (or Never?) Studying Violations of {GDPR's} Explicit Consent in Android Apps. In *Proc. of USENIX Security*, 2021.

[104] Trung Tin Nguyen, Michael Backes, and Ben Stock. Freely given consent? Studying consent notice of third-party tracking and its violations of GDPR in Android apps. In *Proc. of ACM CCS*, 2022.

[105] Erik Nygren. Local IPv6 networks are not addressed / conflating NAT with "Private". https://github.com/WICG/private-network-access/issues/149, 2025.

[106] Okta, Inc. Okta: Secure Identity for Employees, Customers, and AI. https://www.okta.com, 2025.

[107] F. Olano. Google Play Scraper. https://github.com/facundoolano/google-play-scraper, 2015.

[108] Oligo Security. 0-0-0-0-Day: Exploiting Localhost APIs from the Browser. https://www.oligo.security/blog/0-0-0-0-day-exploiting-localhost-apis-from-the-browser, 2024.

[109] Oligo Security. Critical RCE Vulnerability in Anthropic MCP Inspector (CVE-2025-49596). https://www.oligo.security/blog/critical-rce-vulnerability-in-anthropic-mcp-inspector-cve-2025-49596, 2025.

[110] Amogh Pradeep, Muhammad Talha Paracha, Protick Bhowmick, Ali Davanian, Abbas Razaghpanah, Taejoong Chung, Martina Lindorfer, Narseo Vallina-Rodriguez, Dave Levin, and David Choffnes. A comparative analysis of certificate pinning in Android & iOS. In *Proc. of ACM IMC*, 2022.

[111] Ole André Vadla Ravnås. Frida: A world-class dynamic instrumentation toolkit. https://frida.re/, 2025.

[112] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, Phillipa Gill, et al. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. In *Proc. of NDSS*, 2018.

[113] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the Android permissions system. In *Proc. of USENIX Security*, 2019.

[114] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. "Won't somebody think of the children?" examining COPPA compliance at scale. In *Proc. on PETS*, 2018.

[115] Kimberly Ruth, Deepak Kumar, Brandon Wang, Luke Valenta, and Zakir Durumeric. Toppling top lists: Evaluating the accuracy of popular website lists. In *Proc. of ACM IMC*. ACM, 2022.

[116] David Schmidt, Alexander Ponticello, Magdalena Steinböck, Katharina Krombholz, and Martina Lindorfer. Analyzing the iOS Local Network Permission from a Technical and User Perspective. In *Proc. of IEEE S&P Symposium*, 2025.

[117] David Schmidt, Carlotta Tagliaro, Kevin Borgolte, and Martina Lindorfer. IoTflow: Inferring IoT device behavior at scale through static mobile companion app analysis. In *Proc. of ACM CCS*, 2023.

[118] Seon. Fraud Management. https://seon.io/landing/fraud-management, 2025.

[119] Martin Thomson. mDNS access for WebRTC. https://github.com/WICG/local-network-access/issues/22, 2025.

[120] Bill Toulas. ASUS DriverHub flaw let malicious sites run commands with admin rights. https://www.bleepingcomputer.com/news/security/asus-driverhub-flaw-let-malicious-sites-run-commands-with-admin-rights/, 2025.

[121] W3C. Secure Contexts. https://w3c.github.io/webappsec-secure-contexts/#localhost, 2023.

[122] W3C Web Incubator Community Group. Local Network Access Explainer. https://wicg.github.io/local-network-access/, 2025.

[123] W3C Web Security. Same Origin Policy - Web Security. https://www.w3.org/Security/wiki/Same_Origin_Policy, 2025.

[124] W3C WebRTC Working Group. WebRTC: Real-time Communication Between Browsers. W3C Candidate Recommendation Snapshot, 2025. https://www.w3.org/TR/webrtc/.

[125] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond google play: A large-scale comparative study of Chinese Android app markets. In *Proc. of ACM IMC*, 2018.

[126] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, et al. Understanding malicious cross-library data harvesting on Android. In *Proc. of USENIX Security*, 2021.

[127] Qingsi Wang. PSA: Private IP addresses exposed by WebRTC changing to mDNS hostnames. https://groups.google.com/g/discuss-webrtc/c/6stQXi72BEU/m/twDfpwQ4DAAJ, 2019.

[128] Web Application Security Working Group. Secure Contexts. https://w3c.github.io/webappsec-secure-contexts//#localhost, 2023.

[129] WebKit. Tracking Prevention in WebKit. https://webkit.org/tracking-prevention/, 2020.

[130] WebRTC Project. Getting Started with TURN Server. https://webrtc.org/getting-started/turn-server?hl=es-419, 2025.

[131] Nipuna Weerasekara, José Miguel Moreno, Srdjan Matic, Joel Reardon, Juan Tapiador, Narseo Vallina-Rodríguez, et al. Tracking Without Borders: Studying the Role of WebViews in Bridging Mobile and Web Tracking. *Proc. on PETS*, 2025.

[132] Daoyuan Wu, Debin Gao, Rocky K. C. Chang, En He, Eric K. T. Cheng, and Robert H. Dend. Understanding open ports in Android applications: Discovery, diagnosis, and security assessment. *Proc. of NDSS*, 2019.

[133] Yue Xiao, Chaoqi Zhang, Yue Qin, Fares Fahad S Alharbi, Luyi Xing, and Xiaojing Liao. Measuring Compliance Implications of Third-party Libraries Privacy Label Disclosure Guidelines. In *Proc. of ACM CCS*, 2024.

[134] Yandex. Yandex AppMetrica. https://appmetrica.yandex.com/docs/en/, 2025.

[135] Yandex. Yandex Metrica. https://yandex.com/support/metrica/en/, 2025.

Appendices

A Monitored WebRTC Methods

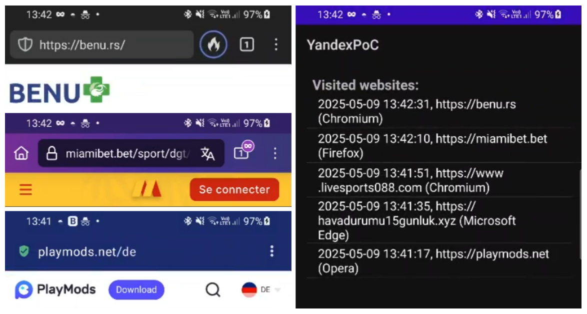
`RTCPeerConnection` (constructor), `createDataChannel`, `addIceCandidate`, `setLocalDescription`, `setRemoteDescription`. All API methods belong to the `RTCPeerConnection` interface [89].

B Yandex Apps

Table 5: Yandex owned apps found listening to localhost ports.

Yandex app	Package name	Tested version
Yandex Maps	`ru.yandex.yandexmaps`	23.5.0
Yandex Navigator	`ru.yandex.yandexnavi`	23.5.0
Yandex Browser	`com.yandex.browser`	25.4.1.100
Yandex Search	`com.yandex.searchapp`	25.41
Metro in Europe — Vienna	`ru.yandex.metro`	3.7.3
Yandex Go: Taxi Food	`ru.yandex.taxi`	5.24.1

C Browsing History Leaks


13:42
https://benu.rs/
13:42
miamibet.bet/sport/dgt/
Se connecter
13:41
playmods.net/de
PlayMods Download

13:42
YandexPoC

Visited websites:
2025-05-09 13:42:31, https://benu.rs (Chromium)
2025-05-09 13:42:10, https://miamibet.bet (Firefox)
2025-05-09 13:41:51, https://www.livesports088.com (Chromium)
2025-05-09 13:41:35, https://havadurumu15gunluk.xyz (Microsoft Edge)
2025-05-09 13:41:17, https://playmods.net (Opera)

Figure 7: Screenshot of our proof-of-concept app. Left are three windows showing websites being visited on three different browsers. Right is our PoC app showing the intercepted browsing history.

D Contacted Yandex Domains

Table 6: Domains contacted by Yandex apps and identifiers observed in network traffic.

Package name	Domain	Identifiers
`com.yandex.browser`	`report.appmetrica.yandex.net`	AAID
`com.yandex.browser`	`startup.mobile.yandex.net`	AAID
`ru.yandex.metro`	`api.browser.yandex.ru`	Build[1]
`ru.yandex.metro`	`report.appmetrica.yandex.net`	AAID
`ru.yandex.metro`	`startup.mobile.yandex.net`	AAID
`ru.yandex.taxi`	`api.browser.yandexcom.net`	Build[1]
`ru.yandex.taxi`	`rb.appmetrica.yango.com`	AAID
`ru.yandex.taxi`	`report.appmetrica.yango.com`	AAID
`ru.yandex.taxi`	`startup.appmetrica.yango.com`	AAID
`ru.yandex.taxi`	`tc.mobile.yandex.net`	MAC[2] Location[3]
`ru.yandex.taxi`	`tc.taxitax.org`	Location[3]

[1] Build: Build fingerprint, [2] MAC: Router MAC address, [3] Location: Coarse and Fine location