Master thesis
Information Science

Radboud University

Transparency or Illusion? Evaluating Apple's iOS Privacy Report and Mobile Tracking Practices

Author:
Max de Groot
s1087160

First supervisor/assessor:
dr. Güneş Acar
g.acar@cs.ru.nl

Second assessor:
prof. mr. Frederik Zuiderveen Borgesius
frederikzb@cs.ru.nl

June 26, 2026

Acknowledgements

I would like to express my sincere gratitude to my supervisor Güneş Acar for his guidance, support, valueable feedback and for always making time to brainstorm with me throughout this project. His expertise and insights were essential to the completion of this work.

Abstract

Apple's iOS App Privacy Report, introduced in iOS 15.2, presents itself as a transparency tool giving users insight into the network activity of installed applications by providing a list of contacted domains, how many times, and at what time each domain was last contacted. However, no prior work has systematically evaluated whether the report accurately and completely reflects the network communications of real-world apps. This paper addresses that gap in two parts.

In the first part, a controlled protocol test evaluates the accuracy of the Privacy Report across ten popular networking protocols - HTTP, HTTPS, WebRTC, WebSocket, MQTT, QUIC, raw TCP, raw UDP and DNS - using a custom test application. Aside from these protocols, iOS's WKWebView, Apple's native interface for embedding web content within an app's user interface, is also covered.

In the protocol test, nearly all contacted domains correctly appeared in the Privacy Report. However, one highly notable exception was found: Domain Name Service (DNS) lookups are entirely absent from the Privacy Report, constituting a functional data-exfiltration channel that is invisible to the Privacy Report, and which leaves users vulnerable to undetected tracking. Furthermore, across all tested protocols, two clear findings emerged from the Privacy Report: connection counts never accurately reflect the true number of network contacts, and timestamps do not reliably update in response to activity.

In the second part of this study, 197 popular iOS applications across 16 App Store categories are dynamically analysed using an automated pipeline built on the WhisperTest framework [1]. Network traffic is captured via packet capture and compared against corresponding Privacy Report entries. Based on the analysis method chosen, 2.4% to 7.5% of all contacted domains was not reflected in the report, the majority of which either shared a parent domain with one already present in the report, or being clear tracking domains. This strongly suggests the domain omissions are genuine report failures rather than background traffic. No evidence of deliberate abuse of the identified DNS data exfiltration channel was found among the 197 applications tested.

These findings support a nuanced verdict: the Privacy Report is technically unfit as a precise auditing tool due to inaccuracies in contacted domains, connection counts, timestamps, and its failure to report DNS-only resolutions. However, given the relatively low domain miss-rate, it does provide ordinary users with a broadly representative overview of application network activity and the (tracking) domains applications communicate with.

Contents

Chapter 1

Introduction

The modern digital economy is heavily supported by the collection and monetization of personal data, a process frequently facilitated through behavioural advertising [2]. On the web, tracking mechanisms such as third-party [3] cookies and browser fingerprinting allow data controllers to construct complex profiles of users, often without their explicit awareness.

The General Data Protection Regulation, an EU law first introduced in 2016, was supposed to give users more control over their personal data. The law stated that personal data could no longer be processed unless one of six legal bases for processing could be provided, one of which is freely given, specific and informed consent from the user [4]. In most circumstances however, for tracking with the goal of behavioural advertising, the only available legal basis for the processing of personal data for behavioural targeting is the data subject's unambiguous consent [5].

However, since the enforcement of the GDPR in 2018, academic research has highlighted a consistent gap between legal requirements and the technical reality. For example, a 2025 study by Martínez et al. on online tracking practices of over one million websites concluded that over half of the researched sites used pixel tracking without the users explicit consent [6].

As companies continue to move their online presence toward apps, these aforementioned privacy challenges shift to mobile devices where the technical infrastructure is even more opaque. For the mobile landscape, there too exists vast literature that demonstrates that GDPR consent compliance is consistently low [7, 8, 9, 10, 11]. However, the existing body of empirical research on mobile GDPR compliance has focused much more so on the Android ecosystem than on the iOS ecosystem.

In its quest to brand itself the 'privacy friendly' mobile device, Apple has released a number of privacy friendly features in recent years, most notable of which are the Ask App Not To Track feature, which lets users choose if app developers get access to unique

identifiers to track them [12], and the Privacy Report, that can give a detailed outline of all domains contacted by individual apps [13].

However, there does not exist any literature that investigates the accuracy and effectiveness of this Privacy Report. Thus, it is unclear whether apps could potentially send data to tracking endpoints without the user's knowledge.

Therefore, the primary research question underpinning this study is as follows:

> To what extent does Apple's iOS Privacy Report accurately reflect the network communications of mobile applications, particularly those involving tracking or behavioural advertising endpoints?

This research question will be answered in two parts. We first develop a test application that uses a variety of networking protocols to communicate tracking information to external endpoints, allowing us to investigate whether certain forms of network communication are by default not reflected in the Privacy Report. Next, we use WhisperTest to conduct a large scale dynamic analysis of iOS apps. This novel tool by Moti et al. [1] allows us to programmatically control an iOS device without jailbreaking it, by using Apple's Voice Control speech input accessibility feature to execute UI actions. Using this framework, we systematically navigate a dataset of 197 popular iOS apps. During navigation, all network traffic is recorded such that contacted endpoints can be compared to the Privacy Report.

By doing this, we hope to answer the following sub-questions:

1. Which network communication protocols available for use by iOS applications are not recorded in Apple's iOS Privacy Report?
2. To what extent does the iOS Privacy Report capture all domains contacted by mobile applications during normal operation?
3. Do mobile applications contact domains that are not displayed in the iOS Privacy Report, and if so, what types of endpoints are omitted?

Overall, our research has three key contributions:

1. **The first empirical evaluation of Apple's iOS Privacy Report** This study provides the first systematic investigation into the accuracy and completeness of Apple's iOS Privacy Report in reflecting the network communications of mobile applications. While previous research has studied mobile tracking and GDPR compliance, no prior work has examined the faithfulness of this specific Apple transparency mechanism.
2. **Protocol-level analysis of Privacy Report coverage** Using a custom-built test application, we evaluate which communication protocols and transmission mechanisms are captured by the Privacy Report, identifying potential technical limitations in Apple's reporting infrastructure.

3. **Large-scale comparison between real network traffic and reported domains** This work introduces a methodology to compare observed network traffic with the domains reported by the iOS Privacy Report, allowing us to quantify discrepancies between actual app behaviour and what is presented to users.

In the remainder of the paper, we first provide the relevant background information and follow up by detailing prior work related to static and dynamic analyses of mobile app privacy, GDPR consent compliance and unconventional forms of tracking. Next, we outline the tests that we have executed to find out whether certain protocols or other specific techniques exist that allow us to communicate tracking information to the web, without the Privacy Report picking up on it. Furthermore, we describe an analysis toolchain based on WhisperTest and apply this toolchain to programmatically navigate our dataset of selected iOS apps. We then analyse this collected network traffic and present our findings. Lastly, we discuss the implications of our findings and conclude by summarizing the results of our study.

Chapter 2

Background

First, we will introduce some relevant concepts and definitions regarding web- and mobile app tracking, data collection and privacy.

2.1 Third-party tracking

Third-party tracking refers to the practice whereby an entity other than the website or application the user intentionally visits - referred to as the "first party" - collects data about that user's online behaviour.

A news website, for example, may embed a third-party advertising script that silently records a user's visit. That same script may equally be present on hundreds of other unrelated platforms, e.g. a shopping site, a health forum or a travel application. This allows companies to identify and track users' behaviour across different websites, and construct a detailed profile of a user's interests, habits and identity, despite them having had no interaction with said company [3].

These profiles are not only stored but are often sold to advertising companies. By analysing user behaviour across multiple websites and platforms, advertisers can gain a deeper understanding of their target audiences' interests and preferences, enabling them to create more relevant advertising content, and are thus able to charge higher prices for advertisements. This practice is commonly referred to as behavioural advertising. For example, behavioural advertising explains why a user who searches for flights on one website may subsequently encounter flight advertisements across entirely unrelated websites and applications [2].

2.2 Consent dialogs

Consent dialogs are user interface elements displayed to users, often when they first visit a website. They are used to ask the user for permission to collect, process or store

personal data, often through cookies or other tracking technologies. Under the GDPR, consent given by a user must be 'freely given, specific, informed, and unambiguous' [4], meaning users must clearly understand what data will be collected and for what purpose before agreeing. However, research has consistently shown that most consent pop-ups are not in compliance with the GDPR [14, 15] due to their use of "dark patterns" [16], deceptive patterns to manipulate the user into giving their consent.

2.3 App analysis types

In this paper, a large scale automated analysis of multiple mobile apps will be conducted. In practice, two main approaches can be taken when analysing third-party tracking in mobile apps: dynamic and static analysis [7].

Dynamic analysis examines application behaviour by executing software within a real mobile operating system environment [7], such as a simulated OS or an actual physical device. A dynamic analysis may be used on the interception and inspection of network traffic, a technique well-suited to identifying data transmitted to third-party services. This method's main advantage lies in its ability to observe actual runtime behaviour rather than merely inferring what a program might do [17]. However, a dynamic analysis is fundamentally constrained by execution coverage: it is impossible to navigate any app to the full extent, and any relevant behaviour that is not triggered during testing will go undetected.

Static analysis, by contrast, decompiles and inspects application binaries without execution, enabling examination of code structure, declared permissions, API calls, and embedded libraries. The most notable advantage here is scalability: pipelines have been demonstrated operating across nearly two million applications [7]. However, static analysis is also prone to false positives, since it examines all code paths without being able to determine whether or not they will be invoked in practice. False negatives can also arise when apps load executable components dynamically at runtime [7], which static inspection cannot detect.

2.4 App Tracking Transparency

Apple's App Tracking Transparency (ATT) framework, introduced in iOS 14.5 in 2021, requires any developer who shares user data with third parties to obtain explicit user consent [12]. The ATT framework allows developers to present users with a standardized prompt, which can be found in Figure 2.1, in which the user can opt to allow the developer to track them, or choose Ask App Not To Track. When a user chooses Ask App Not To Track, the app developer cannot access the system advertising identifier (IDFA), which is often used to track a user across multiple apps [18].

The introduction of ATT had significant consequences for big players in the online advertising industry. In 2022, Meta estimated that Apple's privacy changes would reduce its revenue by approximately $10 billion [19]. External estimates were slightly more conservative, estimating the total industry loss across all companies to be at $10 billion [20].

While ATT is effective at blocking IDFA-based tracking, the framework's protections are narrower than they may appear. Kollnig et al. found that iPhones still transmitted data that could enable device fingerprinting or support cohort-based tracking (tracking at the group level), as only the IDFA is strictly protected at the operating system level [21]. More troublingly, they uncovered evidence of apps "computing and agreeing on a fingerprinting-derived identifier through the use of server-side code", circumventing the ATT framework entirely without the user's knowledge or consent [21].

The validity of the consent collected through ATT has itself been called into question at the regulatory level. In December 2025, the Italian Competition Authority fined Apple €98 million for abuse of a dominant market position, concluding, based on an investigation conducted in coordination with the European Commission, that the ATT prompt fails to meet privacy legislation requirements and effectively forces developers to issue a duplicated consent request for the same underlying purpose [22].

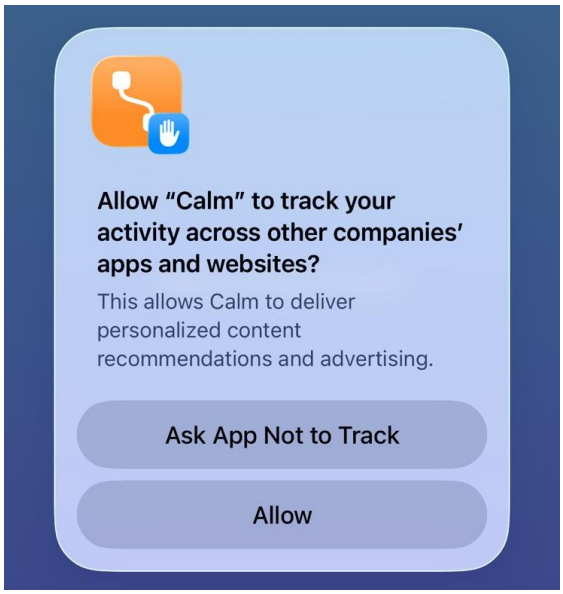


Figure 2.1: Example App Tracking Transparency prompt

2.5 App Privacy Report

Apple introduced the App Privacy Report feature in late 2021 [23] with the release of iOS 15.2 [24] as part of a larger effort to strengthen data transparency for end users. Following the earlier introduction of Privacy Nutrition Labels [25], Apple marketed the Privacy Report as another major privacy-focused improvement [26]. However, unlike many privacy-oriented features that are enabled by default, The App Privacy Report requires users to manually activate it before data collection can begin [13].

The feature allows users to see details about how often apps access their data. Apple divides the feature into two parts:

1. Data & Sensor Access, which allows users to see details about how often apps access their data, like their location, camera and microphone [13].
2. Network Activity, which gives users insight into the domains that have been contacted by each app [13].

This research will focus exclusively on Network Activity.

The App Network Activity "shows domains that have been contacted either directly or from content within an app in the past 7 days" [13]. Apple states "this information also helps provide visibility into domains that may be collecting data about you across different apps and websites" [13]. The feature displays both internal and third-party domains that apps connect to, helping users identify which external services - such as analytics platforms and tracking tools - are contacted behind the scenes [26]. Each app in the privacy report can be clicked on to get a detailed overview of all contacted domains, which can be seen in Figure 2.3. In this detailed view, Apple also differentiates between *Domains contacted directly by this app*, *Websites visited by app*, and *Domains contacted by other content*. It is not clear exactly what method Apple uses to construct the list of contacted domains, as Apple does not publicly document the underlying detection mechanism.

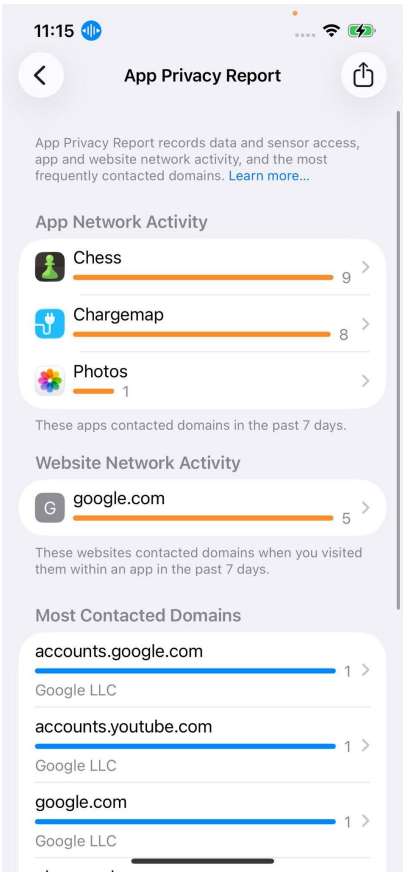


Figure 2.2: Example of the Privacy Report overview

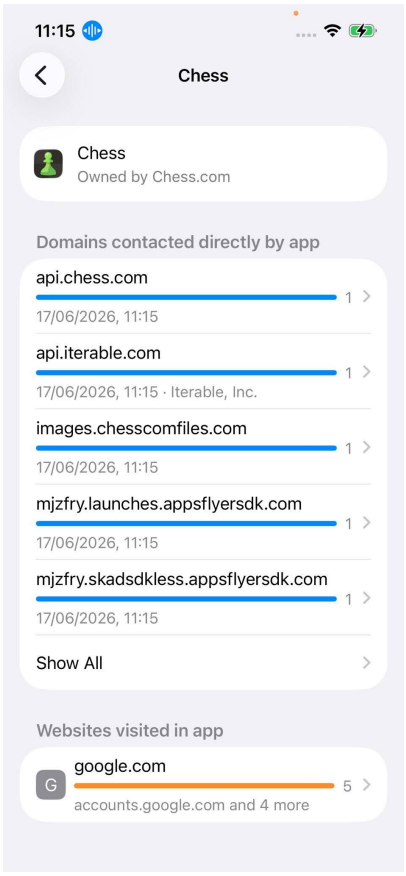


Figure 2.3: Example of the Privacy Report's per-app view

Chapter 3

Related Work

This section will cover related work along with an excerpt of relevant information from each study such as the method and results of the study, and any other relevance to our own work.

Multiple empirical studies have already investigated third-party tracking and potential GDPR-related issues in mobile apps. This related work can be divided into two categories: web related and app related. Since this study is fully app focused, web based related work will not be covered.

In 2021, Kollnig et al. performed an "empirical study into the absence of consent to third-party tracking in Android apps" [11]. They studied a random representative sample of 1,297 free Android apps from the UK Google Play Store using a dynamic analysis. Each application was installed on a test device, left running for 15 seconds without any interaction, and subsequently removed. By analysing the captured network traffic, they analysed which trackers each app used. They found that 71.3% of the 1,201 eligible apps contacted known tracker domains without consent. Another notable result was the amount of Google domains that were contacted: all of the nine most frequently contacted domains were operated by Google, including services linked to its advertising ecosystem. Overall, 58.6% of the apps contacted at least one Google domain [11].

In 2021, Nguyen et al. performed a similar study to that of Kollnig et al. [11]. They studied violations of the GDPR's explicit consent requirement in Android apps [9]. They did so on a much larger scale than ever before, analyzing 86,163 Android apps. Their approach was similar to that of Kollnig et al. [11]: each app was downloaded and run without any interaction, while network traffic is captured. They concluded that 28.8% of all analyzed apps send personal data to data controllers domains without prior consent.

In late 2021, another study by Kollnig et al. [7] investigated the impact of the newly introduced GDPR on third-party tracking in Android apps. To be able to do this at large scale, they used a static analysis to perform the tracking detection. They per-

formed a scan of apps compiled *.dex files to identify URLs used in the app. They then manually cross-referenced URLs that occurred in at least 0.1% of all investigated apps, to determine if the hostname was a tracker. To do this research, they reproduced earlier work by Binns et al. [27] which examined third-party tracking across nearly one million Android apps from 2017, and compared their own data to that of the 2017 study, allowing for longitudinal research. They concluded that in comparison to the 2017 study, the amount of third-party tracking in mobile apps had remained mostly stable, with most apps contuining to integrate third-party tracking [7].

In a 2023 study by Koch et al. called "The {OK} is not enough: A large scale study of consent dialogs in smartphone applications" [10] 3006 Android and 1773 iOS applications were investigated. For iOS they used a jailbroken iPhone and a pre-configured man-in-the-middle proxy to collect traffic. This setup was mirrored on Android. They found that 82.5% of all apps contact a tracking endpoint, with 35.2% of apps sending "at least one request containing a unique identifier rendering the contained information at least pseudonymous and thus covered by the GDPR." Out of all apps tested, only 22.3% of apps displayed any form of consent dialog, and even fewer offered the user "some form of actionable choice" [10]. The researchers ran a follow up analysis, where they performed 350 accepts and 112 rejects on the proper dialogs. They found that when proper dialogs are used, the apps generally respected users' consent choice, with some outliers.

In their 2023 paper, Paci et al. conducted an empirical study investigating third-party tracking compliance in 400 popular mobile applications [8]. Unlike much of the prior literature, which focused exclusively on either Android or iOS, their study examined 200 Android apps alongside their corresponding iOS versions, which made a cross-platform comparison possible. To assess compliance with the ePrivacy Directive and GDPR requirements on valid consent, they developed an analysis template consisting of 16 binary questions spanning six consent requirement. Two researchers independently applied this template to screenshots of each app's cookie banner, reconciling disagreements through discussion. For tracking detection, they used a dynamic analysis. They found that none of the 400 apps fully complied with GDPR and ePrivacy Directive (ePD) requirements, with revocable, specific, and freely given consent being the most commonly violated requirements [8]. Furthermore, they found that nearly half of the analyzed apps (45.75%) continued contacting third-party tracker domains even after users had already explicitly denied consent [8].

In sum, past work clearly demonstrates the widespread prevalence of third-party tracking and data collection in applications on both Android and iOS [7, 8, 9, 10, 11]. Studies show tracking frequently happens without valid user-consent, and therefore likely constitute violations of the GDPR [8, 9, 11]. Therefore, Apple's introduction of the Privacy Report feature is two sided. It has significantly lowered the bar for inspecting app network behavior, allowing ordinary users to observe which external domains an app

communicates with. However, increased accessibility to this information may also encourage users to place excessive trust in the feature. If clear, exploitable gaps in the feature exist, applications would be able to track users without their knowledge without it showing up in the report. They would not have to use any novel tracking techniques but instead could just exploit known vulnerabilities in the Privacy Report feature.

It is also possible that the Privacy Report will be used for academic purposes in the future, as consulting the Privacy Report for the list of contacted domains is far easier than running a network capture, which has been the norm so far [8]. If vulnerabilities in the feature exist, it could contaminate any research that the Privacy Report feature would be used for. Surprisingly, there currently exists no work about the accuracy of the iOS' Privacy Report yet. Paci et al. do call for "an empirical study to analyze the effectiveness of the Privacy Report" [8], but have not performed this research yet.

In 2025, Wang et al published their paper called 'A Big Step Forward? A User-Centric Examination of iOS App Privacy Report and Enhancements' [26] in which they used a focus group of 12 iOS users to identify potential enhancements to App Privacy Report. Despite recognizing the importance of the Privacy Report feature, users identified two usability limitations: insufficient explanations of how their data is used and a lack of clarity regarding the purpose of contacts with external domains. While the researchers did develop practical solutions to realise the aforementioned enhancements, one of which included a Large Language Model (LLM) based pipeline to identify the purpose of any domain contacted by an app, they did not test whether the iOS privacy report actually shows an accurate and complete report of all contacted domains. Furthermore, the study was quite small, with only 46 apps being investigated.

Wu et al. presented their paper 'Transparency or Information Overload? Evaluating Users' Comprehension and Perceptions of the iOS App Privacy Report' at the NDSS Symposium in 2025, having conducted semi-structured interviews with 20 iOS users to evaluate whether Apple's App Privacy Report achieved its transparency goals. While participants understood which apps had accessed their data and sensors, they struggled with the network activity sections, feeling "overwhelmed by the volume of domains contacted and uncertain about what remedial actions they could take" [28]. The researchers recommended labelling domains by purpose and providing aggregated summaries to help users' understanding. However, again, this study did not investigate whether the report provides a complete and accurate account of all domains actually contacted by apps, leaving open the critical question of whether the feature is not only hard to understand but also technically incomplete.

While previously mentioned studies have largely focused on tracking through conventional network requests, recent work has shown that alternative communication mechanisms can also facilitate tracking and data sharing across application boundaries.

A recent case was research by Vlummens et al., who uncovered a novel tracking technique that leverages communication between web content and mobile applications. In their work, "Bridges to Self: Silent Web-to-App Tracking on Mobile via Localhost" they show how third-party JavaScript embedded in websites can use the Hypertext Transfer Protocol (HTTP), WebRTC and Websocket to transmit data to services bound to the loopback interface (e.g., 127.0.0.1) on the user's device [29]. Native mobile apps running background HTTP servers or listening on specific local ports, can then receive these requests and respond with identifiers or other tracking data.

Another popular data exfiltration approach is via the Domain Name Service (DNS) protocol [30]. A DNS lookup is the process of translating a human-readable domain name into an IP address [31]. Since the protocol is not meant for data transmission in the first place, enterprise firewalls often allow DNS traffic to pass without thorough inspection [32]. This allows information to be encoded in DNS subdomain queries and exfiltrated through normal-looking traffic [30].

The most popular approach for DNS-based data exfiltration is DNS tunnelling, where arbitrary data is hidden inside DNS queries and responses, allowing for bi-directional data exchange and relatively high throughput [33]. However, DNS can also be used for data exfiltration in a much simpler form where no bi-directional communication channel is possible and throughput is lower, but simple data can nevertheless be extracted. This makes it an ideal protocol for tracking based use-cases, where only small quantities of data like identifiers need to be extracted, instead of a full bi-directional communication channel. As DNS is also included in this study, we examine the viability of widely known DNS-based exfiltration techniques on iOS.

Chapter 4

Methodology

This research consists of two parts. In the first part, a test application is developed and deployed on iOS to determine whether all network protocols are captured in the iOS privacy report, or whether certain well-known techniques can be performed to make sure specific traffic using said protocols is not reflected in the report. Secondly, building onto the work of Moti et al [1], a set of 240 iOS apps is dynamically analysed to determine whether it is possible for apps to transmit data out of the app and not have it registered in the Privacy Report. Using any potential findings in the first part of the research, we can pay extra attention to the use of certain protocols or methods of data transmission that apps use.

4.1 Protocol test

The first part of this study focuses on determining whether the iOS Privacy Report captures outbound communication correctly across different network protocols, and whether some well-known techniques to evade privacy features work on iOS.

From a technical perspective, the Privacy Report operates on observed outbound network connections, which are then aggregated and presented to the user. Many applications already exist to carefully inspect all network traffic leaving a device, for example WireShark [34]. Such a packet-level inspection of outbound traffic could, along with filtering this traffic by app, be used to generate an accurate Privacy Report. This begs the question: why test individual protocols at all? The reason for this is because the Privacy Report in principle displays the domain network traffic went to, unless an IP address was contacted directly without resolving it first. This is an important distinction, as this means that Apple does not record connections at the this lower, packet-level layer, but rather at a higher level. [35].

To determine how different network protocols are handled by the Privacy Report, a test application is developed for iOS (Figure 4.1) together with a controlled backend environment. The purpose of this setup is to eliminate uncertainty about app behavior

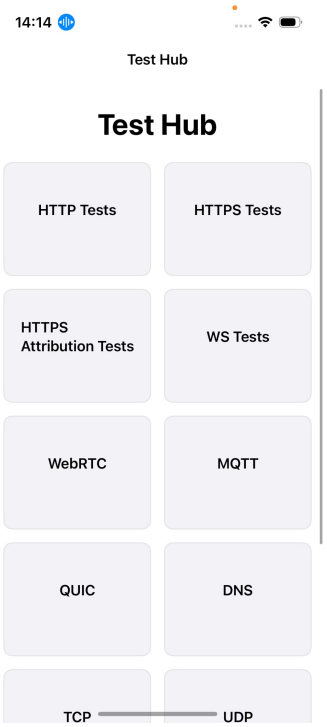


Figure 4.1: iOS Test Application we have developed for this study

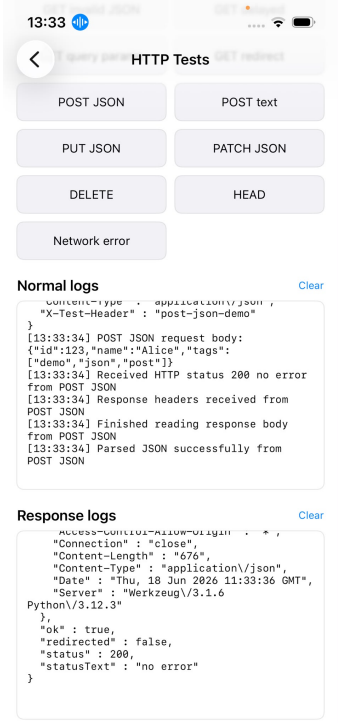


Figure 4.2: Dedicated HTTP tests screen within our developed test application

and to ensure that every observed network contact is deliberately initiated.

The backend services are hosted on one or multiple servers and expose separate endpoints for the iOS test application. The application is configured to contact each endpoint in a controlled and repeatable manner. Because both the client and server are under full experimental control, it is possible to verify with complete certainty whether a connection attempt was made, whether data was transmitted successfully, and whether the contacted domain or endpoint appears in the iOS Privacy Report afterward. Furthermore, a controlled test application isolates specific tests from the complexity of real-world apps. Commercial applications often include many third-party libraries, analytics frameworks, content delivery requests, and background processes, which makes it difficult to determine exactly which transmission produced a given entry in the Privacy Report. In contrast, the custom app can be designed to trigger one communication event at a time, making testing straightforward.

To make the comparison reliable, every protocol will be tested under the same baseline conditions as much as possible. The same device will be used throughout all measure-

ments, and the same server infrastructure will receive all requests. Each protocol test will be repeated multiple times to account for nondeterministic behavior in the operating system, delayed report updates, background execution differences, and possible network failures.

For each run, five observations are recorded.

1. Whether the app successfully established contact with the server.
2. Whether the server received the expected payload.
3. Whether the contacted domain was shown in the iOS Privacy Report.
4. Whether the number of connections was accurate
5. Whether the timing of the connection corresponded with the actual event.

In addition to basic connection success, the protocol test can examine several other factors. One is whether the report behaves differently for short-lived versus persistent connections. For example, a single HTTP request differs substantially from a long-lived WebSocket or MQTT session. Another factor is payload timing: some protocols may transmit immediately after app launch, whereas others may establish a connection first and only send application data later. A third factor is whether secure and insecure variants of the same protocol are reported differently. Testing these variations helps determine whether visibility depends only on the contacted domain or also on the protocol stack and session characteristics.

The protocol tests are also used as an exploratory phase for identifying candidate protocols and transport mechanisms that may warrant further investigation in the large-scale dynamic analysis. Although HTTP(S) traffic is expected to be visible, other protocols may be handled differently by iOS. If a protocol is found to produce confirmed server-side traffic without a corresponding Privacy Report entry, this would indicate a potential blind spot in the reporting mechanism and we can investigate widespread usage of this blind spot in actual apps. Such a result would not by itself prove malicious intent by app developers, but it would demonstrate that the report cannot be assumed to provide a complete overview of all outbound communication.

It is important to note that the list of tested protocols is not exhaustive, nor is the list of tests for each protocol. As previously stated, it merely serves as a way to identify protocols or weak spots that are readily apparent and may warrant further investigation. A complete list of tested protocols and techniques can be found in Table 4.1.

In order to utilize protocols that incorporate encryption, a self-signed certificate will be installed and trusted on the testing device. In a standard production environment, iOS implements stringent certificate validation via its integrated trust store and would categorically reject a self-signed certificate. By manually installing and trusting the

Protocols and techniques
HTTP / HTTPS
WebSocket
WebRTC
MQTT
QUIC (HTTP/3)
Raw TCP
Raw UDP
DNS
WKWebView

Table 4.1: Tested protocols and techniques

certificate through the device's settings, it becomes possible to establish encrypted connections to the custom test server.

For protocols that are not encrypted, App Transport Security (ATS) [36] is deliberately relaxed in the configuration of the test application, as it would otherwise completely obstruct plaintext traffic and hinder the testing of unencrypted protocols. Where possible, both encrypted and unencrypted versions of the same protocol are evaluated. Should the Privacy Report treat them differently, this may indicate that domain logging is sensitive to whether traffic is encrypted or not.

4.1.1 HTTP(S)

HTTP and HTTPS are the most fundamental protocols tested in this study and serve as the baseline against which other protocols are compared. Given that the iOS Privacy Report is primarily designed around web traffic, HTTP and its secure variant HTTPS are expected to appear consistently in the report.

To verify this assumption, HTTP and HTTPS serve as the starting point for protocol testing. Within our dedicated iOS test application, we developed pages to test HTTP and HTTPS connections (Figure 4.2). The backend test server covers a range of cases: GET requests returning different status codes (200, 404, 500), requests with query parameters, server-side redirects, and various request methods including POST, PUT, PATCH, DELETE, and HEAD.

The HTTPS test set largely mirrors the HTTP test set but extends it with a number of scenarios specific to the encrypted transport layer. In addition to these, several response body variants are tested, including plain text responses, intentionally malformed JSON payloads, and artificially delayed responses. The last of these is included to determine

whether the Privacy Report registers a connection at the moment the TLS handshake completes or only once a full response has been received. A set of fault-injection scenarios are also introduced to probe how the Privacy Report handles connections that fail at the transport or TLS layer before any application-level exchange occurs. These include requests to a closed port to trigger a network-level connection error, a scenario in which the server accepts the connection but sends no data, a scenario where plaintext data is sent on a port configured for TLS, and immediate connection teardown by the server after the handshake.

More complex scenarios include a partial TLS handshake in which the server halts mid-negotiation, a scenario where the server reads from the socket before completing the TLS handshake, a deliberately slow TLS handshake, an endpoint configured with an invalid certificate, and an endpoint that responds with HTTP on a port expected to serve HTTPS. These scenarios collectively test whether the Privacy Report registers connection attempts that fail before a full TLS session is established.

Finally, a scenario is included that tests whether data can be covertly transmitted through the TLS handshake itself without a connection ever being fully established. This test is based on a recently disclosed issue in the WICG Local Network Access specification [37], raised in part by members of the localmess research group. The technique exploits the fact that in TLS, the client includes the target hostname in plaintext in the Server Name Indication (SNI) field of the ClientHello message, before any encryption is in place. By encoding a tracking identifier into a subdomain, for example, `uniqueid1234.example.com`, an app can transmit data to a server at the moment of the handshake. If the server extracts the identifier from the SNI field and then immediately closes the connection without completing the TLS handshake, no application-level exchange takes place. The question under investigation is whether the iOS Privacy Report registers such a contact, given that the domain name was transmitted but no connection was formally established. If the report relies on completed connections or application responses as its trigger for logging, SNI-based transmission could represent a channel through which data can leave the device, unrecorded by the Privacy Report.

A problem arises with running a large amount of separate tests. When two requests go to the same domain, it only produces a single Privacy Report entry. Apple merely updates the timestamp at which the domain was most recently contacted. It is not efficient to host a different server for each test we need to run, nor is it efficient to reset the Privacy Report data for each test to make sure it, and not another test, produces a particular entry in the report.

To ensure that each individual request type can be unambiguously identified in the Privacy Report, each endpoint is assigned a unique domain name generated via sslip.io, a wildcard DNS service that maps arbitrary subdomains to a fixed IP address [38]. In practice, this means that if we want to see if HTTP requests that get a 404 status code

response show up in the report, we host an HTTP endpoint on the backend server, whose IP address is, for example, 204.168.169.157. Then, in the client app, we send the request to http://get404response.204.168.169.157.sslip.io/get404response. The iOS Privacy Report does not differentiate between paths, but it does differentiate between subdomains. The request to this subdomain is functionally routed to the same server as all other test requests, but appears as a distinct domain in the Privacy Report, making it straightforward to determine exactly which request types are reflected and which are not.

Attributing network traffic to the user

Any network traffic initated by an iOS app can be attributed to be initiated by either the app itself, or by the user. This can be done by the developer. Attributing certain network traffic to the user can sometimes be useful to app developers, for example for cases where the users has to enter an URL in a navigation bar or web browser [39]. It will be interesting to see whether supposed user-initiated network traffic makes it into the Privacy Report in the same as app-initiated traffic, or at all.

Apple mentions that "all connections, other than those that use SFSafariViewController or ASWebAuthenticationSession, are classified by default as app-initiated." [40] However, a developer can opt to attribute app traffic specifically to the user instead of the app if said network traffic meets any of the following criteria:

1. "User-directed: Network connections to domains that occur only when a user affirmatively chooses to engage with content in your app. For a connection to qualify as user-directed, the user must be provided with a meaningful choice of whether to interact with the content" [40].

2. "Non-primary functionality: The app's features must be functional without this particular network connection" [40].

A vague set of criteria, and one that practically all network traffic to tracking endpoints will meet via the second criterium.

Apple provides a very clear way to attribute network traffic to the user, namely by setting the `attribution` member of the URLRequest object - the standard way of initiating HTTP(S) network traffic - to the value `user`.

We will therefore test this using Apple's provided example code [40]. A test page within our custom test application was developed where requests can be sent without setting the attribution attribute leaving it to its default value, as well as requests that are attributed to the user.

4.1.2 Websocket

Websocket introduce a fundamentally different communication model compared to standard HTTP(S) requests: rather than single requests and responses, Websocket establishes a persistent, full-duplex channel over a single connection. This raises several questions about how the iOS Privacy Report handles such sessions. The core question is whether the report registers only the initial handshake - which begins as an HTTP Upgrade request and is therefore identifiable as a conventional web connection - or whether it also captures the ongoing message exchange over the lifetime of the session.

A baseline test will confirm if the initial WebSocket handshake to a given domain appears in the Privacy Report. Beyond this, several behavioral dimensions will be probed. A long-lived idle connection will be established and kept open for an extended period without sending any messages, to determine whether the mere persistence of an open socket influences how the report records or updates the entry. A low-frequency messaging test will send a single message every few minutes over an otherwise idle connection, examining whether the report reflects continued activity or treats the session as a single event. A high-frequency burst test will send a large number of messages in rapid succession to determine whether message volume or data quantity influences reporting behavior. A late-data test will establish a WebSocket connection and delay data transmission by several minutes after the handshake. Lastly, separate tests are run in which the server or the client immediately close the connection once it's been established.

The secure variant, WSS, will be tested in parallel with WS across all scenarios to determine whether TLS-wrapping the WebSocket session changes visibility in any way, mirroring the HTTP versus HTTPS comparison.

4.1.3 WebRTC

WebRTC differs fundamentally from all other protocols tested in this study in that it is not a simple client-server protocol but a peer-to-peer communication framework. Before any direct data traffic can occur, the two peers must negotiate connection parameters through a signaling process, after which a data channel is established. This architecture, as well as recent abuse by Meta [29], makes WebRTC a particularly interesting candidate for privacy report analysis: the signaling phase and the data transfer phase involve separate network connections to different hosts, and it is not clear which of these contacts, if any, the Privacy Report will register.

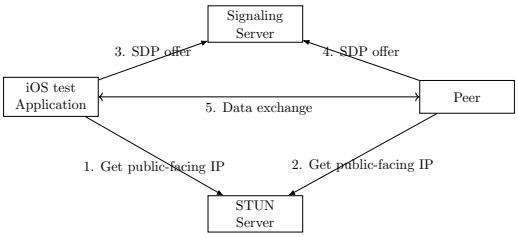


Figure 4.3: WebRTC communication setup process resulting in traffic flowing directly P2P

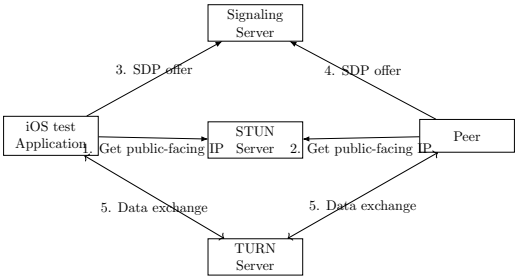


Figure 4.4: WebRTC communication setup process where no P2P connection is possible and traffic is instead routed through a relay

Using WebRTC, two modes of operation are possible. The preferred case is a direct peer-to-peer connection, in which data travels straight between the two peers after they are done negotiating connection parameters (Figure 4.3). Where a direct path cannot be established - for example because of restrictive NAT or firewall configurations - the data is instead carried through a relay [41] (Figure 4.4).

To test both configurations, four separate server components are involved.

1. A self-hosted server that handles the WebRTC signaling phase: when the iOS test application wishes to establish a WebRTC connection with the server peer, it constructs an SDP offer [42] and transmits it via an HTTP POST request to the signaling server, which will then facilitate a direct connection to the peer. The signaling server does not participate in WebRTC itself: no relevant data will ever flow through it. It is merely used by the peers to establish a connection between themselves, or between them and the relay.

2. Google's public Session Traversal Utilities for NAT (STUN) server, which is used by the peers to obtain their public-facing IP address and port.

3. A self-hosted Traversal Using Relays around NAT (TURN) relay server, which acts as a relay for data traffic in cases where a direct peer-to-peer connection cannot

be established due to NAT or firewall constraints.

4. A self-hosted peer server, which acts as the remote endpoint in the WebRTC session, with the iPhone serving as the other peer. Once the peer connection is established, all data exchange occurs over a direct data channel between the iPhone and either the peer- or relay server over UDP, with no further involvement from the signaling server.

To establish a successfull WebRTC connection with the server peer, the iOS device makes at minimum three distinct network contacts: one HTTP contact with the signaling server, one UDP contact with the STUN server, and one UDP contact with either the backend during direct peer connection, or the TURN server in case a relay is needed.

First, we attempt to establish whether or not WebRTC traffic is reflected at all, by establishing a simple connection through using both P2P and TURN separately, and checking which of the contacts appear and which do not. Further tests using a direct P2P connection examine whether the characteristics of the data exchanged over the channel influence reporting at all. These include a high volume burst test, a late-data test in which the first payload is sent 90 seconds after the connection is established, and an interval send test that sends a payload each second, to see whether the Privacy Report timestamp for a single connection is updated correctly, or not.

The central question under investigation is whether the Privacy Report registers all of the servers that are contacted, a subset of servers, or nothing at all. Because the data channel operates directly over low-level UDP, it may be handled differently by the reporting mechanism. If the Privacy Report logs only the middleman contact and not the backend, an application could in principle use WebRTC data channels to transmit large amounts of data to a server that never appears in the report at all, constituting a viable data exfiltration channel.

4.1.4 MQTT

MQTT is a lightweight publish/subscribe network protocol that is ideal for connecting remote devices with a small code footprint, and which uses minimal network bandwidth [43]. It operates over a persistent TCP, TLS or Websocket connection to a central broker, through which clients can publish messages to named topics and subscribe to receive messages from others (Figure 4.5). While MQTT is most commonly used in IoT

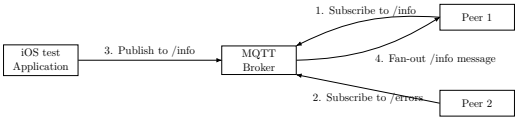


Figure 4.5: Example MQTT publish/subscribe model with topic-based routing via a broker

devices and embedded systems, it has real mobile use cases, for example as a mechanism for push notifications, real-time data feeds, and telemetry reporting. The use of its non-traditional publish/subscribe protocol architecture in iOS apps makes it a relevant candidate for investigation.

A dedicated MQTT broker is deployed on the existing backend infrastructure to serve as the counterpart for the iOS test application. Since, in contrast to WebRTC, MQTT traffic always flows through the broker, there is no need for a peer counterpart to investigate Privacy Report reporting of MQTT traffic.

A set of tests were developed for the iOS test application, which interact with this backend broker. The first and most fundamental test is whether a basic MQTT connection to the broker - the initial TCP connection and CONNECT packet exchange - appears at all in the Privacy Report. This establishes whether iOS recognises MQTT traffic at all. Similar to WebRTC and MQTT, subsequent tests will examine whether the characteristics of the data exchanged over the channel, or the timing, influence reporting.

On top of this, some aspects unique to MQTT are tested. MQTT supports several quality-of-service levels: QoS 0 (at most once), QoS 1 (at least once), and QoS 2 (exactly once) [44]. In a nutshell, these QoS levels are a measure of message delivery reliability. Tests at each QoS level are conducted to determine whether the additional handshake traffic produced by QoS 1 and QoS 2 influences how the session is recorded.

MQTT can be used in unencrypted form (over TCP) and in encrypted form as MQTT over TLS (commonly referred to as MQTTS [45]). Additionally, MQTT can be tunnelled over WebSockets, again both in encrypted and unencrypted forms. All variants will be tested across the scenarios above, as the presence or absence of application layer protocol or encryption may influence whether iOS intercepts and records the connection at the transport layer.

4.1.5 QUIC

Quick UDP Internet Connections (QUIC) is a modern transport protocol developed by Google [46]. QUIC operates over UDP and incorporates TLS 1.3 encryption natively at the transport layer, meaning that connection establishment and encryption are intertwined where in HTTPS, they are sequential. To test this protocol, the dedicated backend endpoint is configured to accept QUIC connections, and the test application initiates requests using HTTP/3, which runs exclusively over QUIC.

The test set covers several unique scenarios. A baseline HTTP/3 GET request will be issued to verify whether a straightforward QUIC-based connection is reflected in the Privacy Report at all. Further tests mostly focus on the characteristics unique to QUIC as a protocol. For example, on iOS, when specifying a host to send a request to, one can pass a parameter called `assumesHTTP3Capable`. Since the fallback behaviour is in-

stigated by iOS itself, it will be interesting to see what iOS will do when we enable this flag for hosts that are in fact not set up for HTTP3 at all, and as such will force iOS to fall back to another protocol. Furthermore, similar to the HTTPS test, we will attempt to exfiltrate data by hiding it in the SNI field, and rejecting the connection altogether.

Together, these tests aim to determine whether QUIC's UDP-based transport and integrated encryption cause it to be handled differently by the iOS network monitoring layer, and whether any of its connection establishment or upgrade mechanisms represent potential blind spots in the Privacy Report's coverage.

4.1.6 Raw TCP

As all previously described tests are primarily focused on whether the iOS Privacy Report records communication using higher-level networking protocols, it would also be interesting to see what happens when no high-level protocol is used at all. Raw TCP testing strips away any overlying protocol such as HTTP or TLS and instead sends arbitrary byte sequences directly over a TCP connection to a controlled server.

The raw TCP test set covers several scenarios, including both success and failure paths, as well as tests targeting the timing and connection-count features of the Privacy Report.

If raw TCP connections are not reflected in the Privacy Report, this would indicate that the reporting mechanism depends on application-layer protocol recognition rather than transport-layer activity. This would have implications for the large-scale dynamic analysis phase, as any app transmitting data over a non-standard or proprietary binary protocol over TCP could potentially do so without generating a corresponding report entry.

4.1.7 Raw UDP

UDP, another low-level protocol, differs fundamentally from TCP in the fact that it is connectionless. There is no handshake, no guaranteed delivery, and no session state [47]. Because a UDP datagram can be dispatched to an IP without any prior exchange, it represents one of the most minimal forms of network communication possible, and it is unclear whether such transmissions fall within the scope of what the Privacy Report is designed to capture.

UDP-unique tests include oversized fragments subject to fragmentation on the IP layer, as well failure scenarios like sending UDP datagrams to ports with no active listener, causing ICMP port-unreachable errors.

Given that UDP carries no handshake and no session, the Privacy Report might fail to register these transmissions entirely. If confirmed, this would suggest that an app

using UDP-based exfiltration could transmit data without any corresponding entry appearing in the report.

4.1.8 DNS

DNS resolution occupies an unusual position in the network stack relative to the other protocols tested in this study. Every connection to a named host begins with a DNS lookup, meaning that DNS activity is in some sense a prerequisite for all other traffic rather than a distinct communication channel. However, as previously described [30, 32, 33], DNS can also be used as a data transmission mechanism in its own right, entirely independently of any subsequent connection. By encoding information into the subdomain portion of a query, an application can cause data to leave the device at the moment of resolution, without ever establishing a TCP or UDP connection to the resolved address.

To test whether or not the Privacy Report is vulnerable to DNS based data-exfiltration, we make use of Canary Tokens, a service specifically designed for detecting and logging DNS-based interactions. Canary Tokens provides unique hostnames that, when resolved, trigger a callback to the service's infrastructure and log the resolution event along with any data encoded in the subdomain [48].

Importantly, the service also supports encoding arbitrary data directly into the subdomain of the query, meaning that a lookup for a hostname such as `secretdata.uniquetoken.canarytokens.org` both transmits the encoded value to the Canary Tokens server and serves as a verifiable record that the resolution occurred. This makes it straightforward to confirm server-side receipt of the query independently of anything the iOS Privacy Report may or may not record.

If DNS resolutions that are not followed by a completed connection are absent from the iOS Privacy Report, this would indicate a potentially significant gap in the transparency mechanism. Each DNS packet may be limited in size, and thus in the amount of data that can be exfiltrated, but many DNS packets can be sent, which would enable an application to transmit identifiers, telemetry, or other data without any corresponding entry ever appearing in the report. Given how straightforward this technique is to implement and how difficult it would be for a user to detect through ordinary means, a negative finding here would be among the more practically significant results of the protocol test phase.

4.1.9 WKWebView

WKWebView is Apple's supported API for embedding and rendering web content inside third-party iOS applications. Applications commonly use WKWebView to load remotely hosted content, serve advertisements or simplify a login process.

WKWebView is one of the few techniques or protocols that Apple divulges information about in relation to traffic originating from it showing up in the Privacy Report: "App Privacy Report doesn't include network activity from private browsing sessions in browser apps." However, "Network activity and websites visited in non-browser apps that provide private modes are displayed in App Privacy Report" [13]. Based on this, we expect traffic originating from a WKWebview to be correctly reflected in the Privacy Report.

For each protocol section in this study, a manually selected subset of the most informative test cases will be replicated inside a WKWebView and the results compared against their native counterparts. The selection will prioritise tests that produced notable or ambiguous results in their native contexts, as well as tests that were not technically feasible to run without a browser-based execution environment, like iFrame loads from a webpage. Implementing a private browsing mode in a WKWebView, which is possible according to Apple's documentation [49], will also be tested to see if Apple's claim holds up.

Not every test case can be reproduced inside a WKWebView. For example, raw TCP and UDP transmissions are not accessible from a web context, and deliberately initiating certain TLS faults depend on low-level socket control that the WebView environment does not expose. The precise set of tests that will be executed inside the WKWebView cannot be determined in advance, as the selection depends in part on findings made during data collection. Tests that result in interesting findings in the native protocol phase will be prioritised for replication in the WKWebView context, making this a partially adaptive component of the methodology.

4.2 Dynamic analysis

Aside from checking whether network traffic by certain protocols are not reflected on the Privacy Report, we will also conduct a dynamic analysis on a set of popular iOS applications. By doing this, we check how the Privacy Report performs for real-world applications, as well as check whether some apps have found more ways to hide their activities from the Privacy Report. Apps will be launched and automatically navigated to trigger as much network traffic as possible. All network traffic is recorded, so it can be compared to the Privacy Report afterwards.

4.2.1 WhisperTest and pymobiledevice3

The core of the technical pipeline is based on the WhisperTest framework by Moti et al. [1]. Unlike traditional tools that rely on jailbroken environments or limited UI automation frameworks, WhisperTest leverages Apple's Voice Control accessibility feature to simulate realistic user behavior using voice commands. This allows the system to navigate complex interfaces, trigger in-app functionality, and interact with buttons,

consent dialogs and login screens. For some functionality, WhisperTest makes use of the Python module pymobiledevice3. pymobiledevice3 is a pure Python 3 implementation for interacting with iOS devices [50]. It implements the device-side protocols entirely in Python [51]. It exposes a broad set of capabilities, including profile and application management, network sniffing via Packet Capture (PCAP), as well as support for syslog streaming, which allows the pipeline to capture live system logs from the device during test execution [52]. This is valuable for observing runtime behavior, diagnosing crashes, and verifying that app actions triggered by the voice command automation produce the expected system-level responses.

4.2.2 App selection and acquisition

To ensure a broad testing range, apps from each category of the Dutch iOS app store will be tested. Luckily, Apple provides a neat overview of all app categories, as well as the most popular apps per category [53]. In its overview, Apple differentiates between 'Apps' and 'Games'. Both Apps and Games have 15 sub-categories. We have chosen to include all 15 'Apps' categories, but only one 'Game' category. For each category, Apple has selected a number of 'essential' apps. These are popular apps, highlighted by Apple. For each category, Apple has highlighted anywhere between 13 and 15 apps.

To populate its web page with the correct content, Apple uses a public internal API endpoint to fetch data. We are able to use this endpoint to collect data about the app's bundle identifiers, an identifier that is unique per app and that we can use to automatically download app onto the test device. Selecting all highlighted apps in each of the 16 selected categories, and filtering out duplicates and paid apps, leaves 197 apps to be tested across 16 categories. The number of apps tested per category can be found in Appendix A.

4.2.3 DNS cache clearing

During the dynamic analysis, a packet capture (PCAP) will be used to track all network traffic generated by the iPhone. During a PCAP, data packets, the units of data and payload that make up all network traffic, are intercepted [54]. However, these packets are not identifiable by domain, the unit that the Privacy Report produces. Instead, data packets contain IP addresses.

To resolve an IP address back to its corresponding domain, we rely on DNS lookups: when an app contacts a domain, the device needs to know which IP address belongs to said domain in order to start communicating with it [31]. It does this by issuing a DNS query. The response of this query - the IP belonging to this domain - is captured in the PCAP.

However, a problem can arise: many different apps may contact, for example, `facebook.com`, and the OS may want to avoid having to re-query this domain each time. The de-

vice may therefore store the result of the query in its DNS cache [55], and save it for an extended period of time. In this case, no DNS query is issued during the recording, leaving us with an IP address we cannot reliably map back to a domain name. If this scenario when comparing the PCAP to the Privacy Report, we would conclude that traffic went to some unknown IP address, while the Privacy Report might conclude that traffic went to some domain that we did not see being resolved in the PCAP. In this scenario, the Privacy Report paints a true picture, but that's not what an analysis would conclude.

To prevent this, the device's DNS cache needs to be cleared before each app is tested. This ensures that every domain contacted during the test session triggers a fresh DNS lookup, producing a complete and accurate mapping between IP addresses and domain names that can be cross-referenced with the domains reported in the Privacy Report. Apple unfortunately does not offer a standard way to reset the DNS cache. However, several online sources [56, 57] mention toggling the Airplane mode on and off and waiting 5-10 seconds in between as a viable option to clear the iPhones DNS cache. Therefore, we developed an application to test this theory.

This simple test application is able to send a simple HTTPS request to any URL of choice. Using the test application, the same GET request was sent to the same domain three times in a row. During the first test, the Airplane mode was not reset in between requests. The endpoint of choice is resolved only one time, as expected (Figure 4.6). During the second test, when the Airplane mode is reset in between requests, the endpoint of choice is resolved three times, which confirms the local DNS cache is reset in between each request (Figure 4.7).

However, this method of identifying contacted domains using DNS queries is not foolproof: it is only possible if the DNS queries are sent over plain, unencrypted UDP. It is also possible to send DNS queries over HTTPS, TLS and QUIC, which means that DNS data will be encrypted. If any apps do this, we will not be able to see a hostname be resolved to an IP address.

4.2.4 Custom voice commands

To ensure fast and reliable execution of some standard operations during app navigation, custom iOS voice commands will be used. Custom voice commands allow us to record a series of sequential voice commands that can be played back with a single named command [58]. This benefits the research greatly because it increases the reliability that a command will be executed successfully. These custom commands include opening, saving and clearing the privacy report, starting and stopping screen recording and turning Airplane mode on and off. By monitoring the system logs of the iPhone, we can detect whether or not a custom voice command was heard by the iPhone. Voice commands are typically retried up to three times.

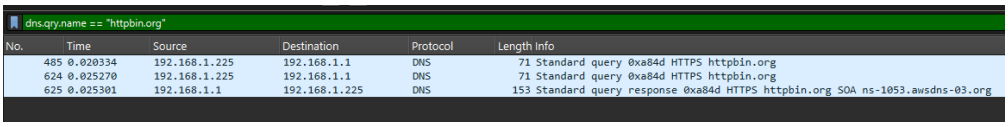


Figure 4.6: DNS Traffic to test endpoint when cache is not reset

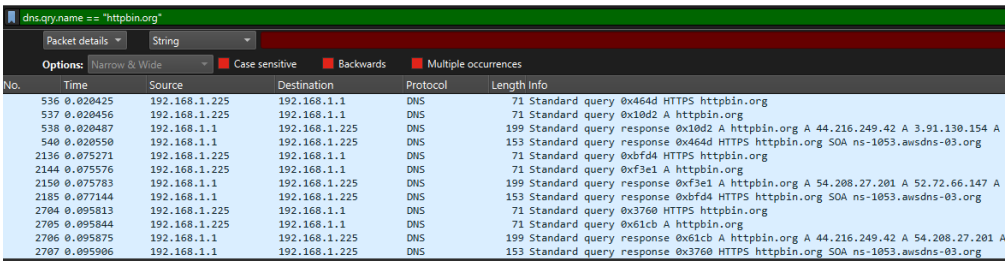


Figure 4.7: DNS Traffic to test endpoint when cache is reset

4.2.5 Automatic navigation

Due to the diversity of UI designs, app navigation is a complex task which cannot be solved solely by a rule-based algorithm. Therefore, WhispertTest relies on a number tools and algorithms in order to constantly devise the next best action, given the current screen.

Screen element extraction

To determine the best next action, WhisperTest first identifies what elements are currently present on the screen. To do this, one of two methods is used.

1. Elements which have been given a predefined Accessibility label are extracted directly from the iPhone. Accessibility labels are normally used in order to make an iPhone useable for, for example, blind people. They are hidden, textual labels accurately describing each element on the screen. Since we can extract the accessibility labels directly from the device, this method of extracting screen elements is the faster one. However, in practice, not all developers give their content accessibility labels, making this method primarily useful for quickly identifying native dialogs like ATT and other iOS prompts.

2. A screenshot is made and visible screen elements are extracted using Optical Character Recognition by OmniParser, a screen parsing tool made by Microsoft [59]. This method is slower, but does not require accessibility labels to be present. A difference in the result of both element extraction algorithms given the same screen can be found in Appendix G]

Decision algorithms

The navigation pipeline uses five different algorithms each aimed at trying to devise the best next action to navigate the app, with the goal of covering as much of the app as possible, given the current screen and a history of previously attempted actions. These are the following:

1. Accessibility labels are extracted, and a rule-based algorithm attempts to devise the next best action.

2. Screen elements are extracted using OmniParser, and the aforementioned rule-based algorithm attempts to devise the next best action.

3. Accessibility labels are extracted, and used to prompt an LLM. The LLM is instructed to come up with the most logical next command based on the available screen captions and the action history. An example prompt can be found in Appendix C.

4. Screen elements are extracted using OmniParser, and used to prompt an LLM. The LLM is instructed to come up with the most logical next command based on the available screen captions and the action history.

5. A screenshot is made and an LLM is prompted to select the most logical next command, directly given the screenshot. Screen elements are not extracted beforehand.

Full navigation pipeline

For our navigation pipeline, we did not make any major changes compared to the original WhisperTest research. However, our navigation pipeline differs from the pipeline described by Moti et al. [1] on two points:

1. While the original data collection pipeline only tries to resolve Apple dialogs such as ATT prompts using a rule-based approach once per navigation iteration before switching to an OCR or LLM-based approach, we instead try to resolve all Apple dialogs until none remain, and also check for new dialogs between each OCR- and LLM-based navigation attempt.

2. Since we are working with a relatively weak vision model due to hardware constraints, the image based decision algorithm is only used as a last option, if all other navigation attempts have failed.

Having applied these changes, pseudocode of the automatic navigation process looks as follows:

Algorithm 1 Pseudocode of automatic navigation loop

```
while true do
    resolve_all_system_dialogs_using_accessibility_data()

    if navigation time is up
        break
    end if

    if rule_based_ocr_approach() is successful
        continue
    end if

    resolve_all_system_dialogs_using_accessibility_data()

    if llm_based_accessibility_approach() is successful
        continue
    end if

    resolve_all_system_dialogs_using_accessibility_data()

    if llm_based_ocr_approach() is successful
        continue
    end if

    resolve_all_system_dialogs_using_accessibility_data()

    llm_based_vision_approach()
end while
```

WhisperTest allows for a choice in which LLM models will be used for navigation-decision making, based on the hardware and software limitations of the system it is used on. For our research, we used llama3 8B [60] for all pure text prompts, and gemma3 4B [61] for all image-based queries. For all LLM prompts, a temperature - a parameter which controls the randomness of text that is generated by LLMs [62] - of 0.0 is used for maximum determinism.

4.2.6 'Accept' mode

WhisperTest can be run in a so-called 'reject' mode, in which all consent dialogs are rejected, or 'accept' mode, in which they are accepted [1]. To ensure as much network traffic and tracking is initiated as possible, we make use of the accept mode. During normal operation, WhisperTest constantly tries to find both native consent dialogs (Ask App Not to Track) as well as non-native dialogs (cookie banners). Both of these forms

of consent dialogs need to be handled correctly.

ATT and other iOS prompts are fully standardized, so they are easily caught using accessibility labels and the rule-based algorithm. To detect if a non-native consent dialog is present on the current screen, WhisperTest extracts screen content and analyzes it with text-based LLM (Appendix D). When a consent dialog is detected, the system dynamically updates the next LLM prompt with the appropriate navigation instructions [1].

4.2.7 Data collection pipeline

For navigation of the applications, a pipeline that is derived from the original WhisperTest paper [1] is used, with some minor changes. For each of the 197 selected applications, the same data collection pipeline is run:

1. The iOS privacy report is cleared using a custom voice command.
2. Airplane mode is turned on using a custom voice command.
3. The PCAP is started.
4. The app is installed.
5. Airplane mode is turned off using a custom voice command.
6. Automatic navigation (see subsection 4.2.5) of the application for a period of four minutes ensues.
7. The PCAP is stopped and saved.
8. The Privacy Report is saved, using a custom voice command.

During data collection, both iCloud Relay [63] and private WiFi address [64] are kept turned off to ensure that data goes directly to the specified hosts, and isn't rerouted through Apple's relay, which would contaminate data collection.

Furthermore, to reduce background network traffic as much as possible, data collection is performed on an iPhone with as few installed apps as possible. Only a handful of Apple apps remain installed, as they cannot be removed.

4.3 Data analysis

Once all data collection is complete, a thorough data analysis is performed. In this analysis, the Privacy Report data is extracted and filtered for network activity by the app that was actually tested. Next, the PCAP is analysed to extract the contacted domains. These two datasets are then compared to identify overlaps and discrepancies between the reported and observed network communications. However, there are some factors to take into account while analysing the data.

4.3.1 Domains and IPs

Firstly, the Privacy Report reports two types of data. As previously mentioned, the report contains the 'domains contacted' by an application. However, when an application directly contacts a raw IP address instead of a resolvable domain, this IP address is also reflected in the Privacy Report. This introduces an analysis problem.

If an application first sends an HTTP request to a domain, and subsequently sends some completely unrelated TCP traffic **directly** to the raw IP address behind this same domain, the Privacy Report will correctly report this as two different network contracts: one where the domain was contacted, and one where the IP address was contacted. However, while performing an analysis of the PCAP, there is no way of knowing whether the subsequent TCP traffic was or was not a part of the HTTP request. Therefore, we have to account for this in our data analysis, which is fortunately trivial: if the Privacy Report reports contacts to raw IP addresses in addition to domain names, we simply check whether those IP addresses occurred in the PCAP.

4.3.2 Packet filtering

Even though nearly all apps have been removed from the iPhone, background traffic can still occur from these apps, or from other OS processes running on the iPhone. The pymobiledevice3 module makes use of iOS's remote virtual interface (RVI) function [65] in order to expose a Python service capable of capturing iOS device network traffic. Luckily, Apple adds two attributes to each individual data-packet that it sends over this RVI: the Process Identifier (PID), and Effective Process Identifier (EPID) of the app that initated the data-packet. Using this function, we could in theory filter network traffic by the app we're currently testing, and achieve zero visible background traffic.

However, after manually testing this feature thoroughly, we discovered some traffic sent by the app itself is stripped of its PID identifier before being sent to the monitoring device. This makes the filter miss a lot of traffic that the app actually did initiate. For example, some apps spawn a WKWebView and make users log in on this page, instead of logging in within the app itself. The requests made by these in-app webpages are missed by the PID filter. We therefore opt not to use this filter.

4.3.3 Identifying Apple domains

Since the PID filtering feature cannot be used, we need some way of filtering out background traffic so we do not detect any false positives when comparing the Privacy Report to the PCAP. Several obvious methods exist, such as filtering out all traffic to `apple.com` or Apple-owned domains (such as `icloud.com`, `aaplimg.com`), as well as filtering out traffic to any IP address starting with 17., as Apple owns them all [66].

We initially expected the aforementioned DNS reset function to also concern all Apple

domains. However, while during the manual test many Apple domains were re-resolved, some domains, presumably those necessary only for background operations, were not. Because of this, a situation might occur where certain Apple owned domains are not re-resolved during a DNS reset. As a result, we might see traffic to an Apple IP address without seeing the accompanying DNS query, and could therefore not reliably identify the domain as an Apple domain.

To counter this problem, we have opted to include a preprocessing step for the data analysis. In this preprocessing step, all PCAP analysis files are first processed to identify any Apple domains and IP addresses, which are then saved to a separate file. Since data collection occurs over the course of several days, it is plausible that most, if not all of Apple's background domains were resolved at least once. Using this preprocessing step, when encountering unknown raw IP addresses during the final data analysis, we can very easily check whether or not a certain IP address was an Apple domain or not. If it is, it is filtered out as background noise.

Unfortunately, this approach still does not filter out all background noise. During normal operation, Apple might fetch some data in the background (e.g. some images or notes) which is stored in the cloud. For efficiency, this data might be fed through an external Content Delivery Network (CDN), not Apple's own servers. In this case, there is no way of knowing whether it was the app, or some background service, that initiated the traffic.

4.3.4 Domain aggregation heuristic

Most likely in an attempt to make the Privacy Report more user-friendly, Apple decided to aggregate domains of which it can identify a *domain owner*. For example, if Apple sees network traffic to the domain `logs.ads.vungle.com`, Apple truncates this domain to `ads.vungle.com` in its Privacy Report entry, and displays the domain's owner, Vungle Inc, in the UI (Figure 4.8). Accounting for this heuristic approach in our data analysis might result in false negatives, since if two or more sub-domains of the same domain owner occur, we cannot accurately state whether Apple missed one or not. Since we are interested in what effect including this approach has on the analysis results, we will account for this heuristic in our data analysis, in a separate step.

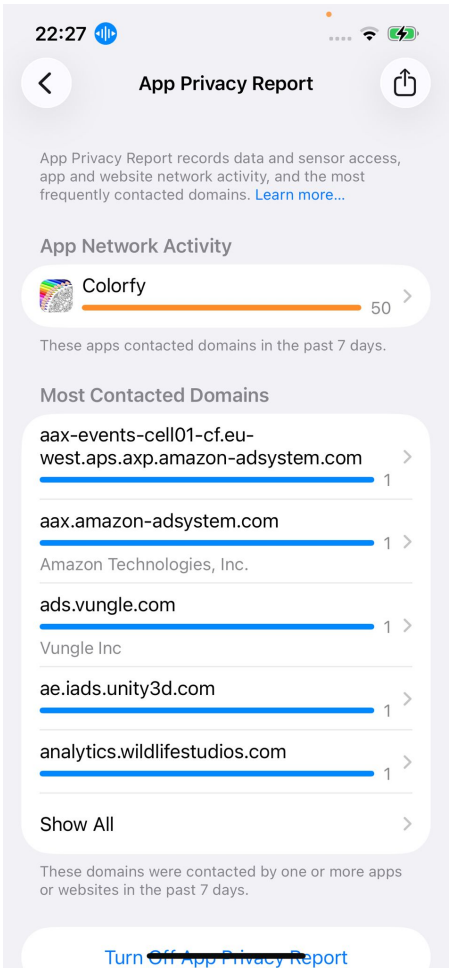


Figure 4.8: Due to Apple's domain aggregation heuristic, domains `config.ads.vungle.com` and `logging.ads.vungle.com`, which were seen in the PCAP, are both represented by a single entry in the Privacy Report: `ads.vungle.com`

Chapter 5

Results

5.1 Protocol tests

This section reports the findings of the protocol tests described in the methodology. For each protocol, the results are summarised in terms of the five observations defined in the methodology: whether the application successfully contacted the server, whether the server received the expected payload, whether the contacted domain appeared in the iOS Privacy Report, and whether the timing and number of connections recorded in the report corresponded to the actual events.

All code used in the tests have been open-sourced, though identifying information has been removed [67].

5.1.1 Cross-cutting findings

Across the test set as a whole, two patterns emerged that are largely independent of specific protocols. These are discussed first to avoid repeating them in every subsequent section. The first concerns the connection count displayed by the Privacy Report. In none of the tests, across any protocol, did the number of connections shown in the report accurately correspond to the number of connections actually initiated or established by the test application. When a test application opened multiple connections to the same domain, the Privacy Report continued to display a connection count that did not increment in line with the actual network activity. Most often, the connection count just remained at one. This behaviour persisted even when the application was force-closed and relaunched between tests, and even when the device's DNS cache was reset. The report therefore clearly cannot be relied upon as an accurate measure of how many times an app has contacted a given host; it functions, at best, as a binary indicator that contact has occurred at some point.

The second finding concerns the timestamp associated with each domain entry. When multiple connections were made to the same domain, the timestamp shown in the Pri-

vacy Report was not updated to reflect the most recent connection. In repeated tests, the displayed time clearly corresponded to an earlier connection, not to the one most recently observed on the server. The timestamp did update under some conditions, most consistently when several minutes had elapsed between connections, but updates were neither immediate nor reliable. As with connection counts, the timestamp field of the Privacy Report should therefore be treated as an approximation.

5.1.2 HTTP(S)

The HTTP and HTTPS tests, which can be found in Appendix E.1 and E.2, served as a baseline for all other tests. HTTP(S) traffic was very strongly expected to reliably show up in the Privacy Report, and this expectation was met.

A total of 41 test cases across HTTP and HTTPS were run. These tests included GET requests with several different status code responses, malformed JSON, delayed responses as well as fault-injection scenarios like connections to unreachable endpoints, partial TLS handshakes, reads before TLS, slow TLS handshakes, invalid certificates and HTTP responses on HTTPS requests: in every case the domain was logged in the Privacy Report regardless of whether a TLS session was ever fully established or any application-level data exchanged.

The custom SNI scenario, which probed whether data encoded into the Server Name Indication field of a ClientHello can be transmitted without producing a Privacy Report entry, also resulted in the targeted subdomain appearing in the report. Even for requests that were attempted on closed ports, where there was no listening server, the targeted subdomain appeared in the Privacy Report. The same was true for tests where the server accepts the connection but never responds. This indicates that for HTTPS-style traffic, the threshold for inclusion in the Privacy Report is reached well before any complete encrypted session, and possibly at the level of the initial DNS resolution or TCP handshake.

Attributing network traffic to the user The test using Apple's provided example code, which uses a simple URLRequest object and sets the attribution member to *user*, was also performed. Initially, the requests were found not to show up at all. When initiating two requests, one to `appsflyersdk.com` attributed to the app, and the other to `facebook.com` attributed to the user, the Privacy Report would only report the request to `appsflyersdk.com` in the main screen (Figure 5.1). This is supported by the fact that the total connection count for the app is one. Only when clicking on the initiating app in the Privacy Report, to get a more detailed page of the apps network traffic, does the request to `facebook.com` show up (Figure 5.2). While this does not constitute a privacy vulnerability, since there is no way of hiding the contacted domain altogether, it does provide developers with the opportunity to obfuscate requests to tracking domains to some extent.

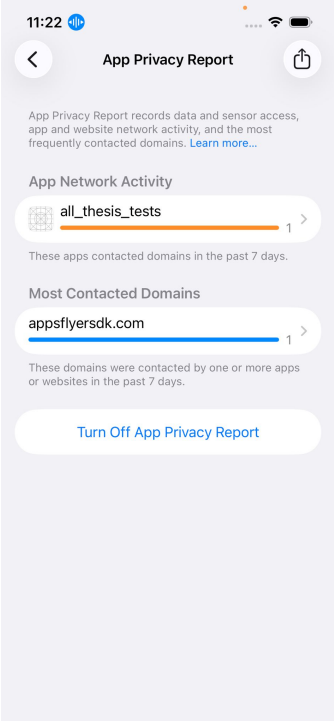


Figure 5.1: Privacy Report overview after request attribution test

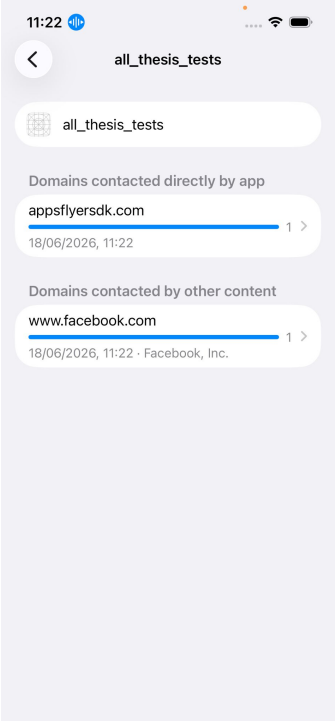


Figure 5.2: Privacy Report per-app view after request attribution test

5.1.3 Websocket

The Websocket test set, which can be found in Appendix E.4, covered seven unique scenarios across both the plaintext and encrypted versions of the Websocket protocol. These tests included baseline handshake behaviour, idle long-lived connections, low-frequency messaging, high-volume bursts, late-data transmission, server-initiated closes and client-initiated closes.

In all 14 cases the test application successfully contacted the server, the server received the expected payload where one was sent, and the contacted subdomain appeared in the iOS Privacy Report. The cross-cutting issues were observed here as well, and are particularly visible in the WebSocket context because the protocol's full-duplex nature offers many opportunities for activity that the report fails to reflect. Sending many messages over an existing WebSocket connection, both low and high frequency, does not update the timestamp in the Privacy Report, even though the server does record every message correctly. Closing and reopening the connection within a short time window also fails to refresh the entry. This clearly shows that for Websocket sessions, the Privacy Report essentially treats the entire lifetime of a session, from the initial handshake until the

connection closes, as a single event. It offers no indication of how long the session lasted, when data was last exchanged, and how many messages were exchanged, even though Apple's UI does allow for this. The contacted domain is recorded, but session-level activity is not.

5.1.4 WebRTC

The WebRTC test set was structured around the two principal modes of WebRTC operation: a direct peer-to-peer connection facilitated only by signalling and STUN, and a relayed connection routed through a TURN server. All six scenarios, which can be found in Appendix E.3, produced correct Privacy Report entries for the contacted domains. In the peer-to-peer scenarios, the device's contact with the STUN server, the signalling server and the peer endpoint were all captured in the report. In the TURN scenario, the relay server appeared as well. Server contact and payload reception were confirmed in all cases.

5.1.5 MQTT

MQTT was tested across 12 unique scenarios, which can be found in appendix E.5. All tests were run using four different underlying protocols: TCP, TLS, WS and WSS. These tests included connect-only sessions, idle persistent sessions, brief connect-disconnect sequences, single and burst publishes, low-frequency publishing, subscribe-only, all three QoS levels, retained messages, and a late-data variant. In all tested combinations the broker domain appeared in the Privacy Report. This also holds for every QoS level despite the differences in control packet exchange between QoS 0, QoS 1 and QoS 2. The use of different underlying protocols to perform the tests had no effect on the results.

5.1.6 QUIC

The QUIC test set, which can be found in Appendix E.6, consisted of a total of six scenarios. Tests started with a baseline HTTP GET, where a simple GET request is sent with the flag `assumesHTTP3` set to true, leaving the OS to figure out how it wants to send the request. Based on the server logs, we can see that iOS did choose to send the request over QUIC. Other tests include a forced QUIC-only configuration, and a blocked-UDP-port scenario forcing iOS to fall back to TCP. All traffic was found to correctly show up in the Privacy Report.

5.1.7 Raw TCP

The raw TCP tests, which can be found in Appendix E.8, moves away from higher-layer protocol tests and probes whether the Privacy Report logs a connection that never uses HTTP, TLS, or any other recognisable protocol on top of TCP. In all five cases the contacted subdomain appeared in the iOS Privacy Report.

This result shows that even connections that never exchange HTTP, TLS, or any other high-level protocol are recorded. The immediate-close and connect-with-no-data tests are the most telling, in that they show that the threshold for inclusion does not require any payload exchange at all - a successful TCP handshake to a named host is enough. As with all other protocols, the report does not reflect repeated transmissions over a long-running connection; the long-running-stream test produced one entry, not a stream of updates.

5.1.8 Raw UDP

The UDP test set, which can be found in Appendix E.9, probes an even more minimal form of outbound network communication: a connectionless UDP datagram dispatched directly to a raw IP address. Testing raw UDP answers the question: does a connection need to be established before a Privacy Report entry is displayed, or is the mere attempt enough?

In all of the six scenarios the targeted subdomain appeared in the Privacy Report. This includes the scenario where data was sent to a port with no listener in which no acknowledgement of any kind is returned to the sender, and the scenario in which the datagram is oversized and thus split across multiple IP fragments. The result indicates that, for the purposes of the Privacy Report, raw UDP transmissions to named hosts are recorded much like other protocols at the level of domain visibility, even when no connection is ever established. The connection-count and timestamp limitations do apply here, the same as elsewhere: a stream of datagrams sent over five minutes does not produce a stream of updates in the report.

5.1.9 DNS

The DNS scenario tested whether a hostname resolution that is not followed by any subsequent connection is recorded in the iOS Privacy Report. The test application performed a DNS lookup for a unique Canary Tokens subdomain. Using the provided documentation [48], additional data such as an arbitrary identifier, the name and model, as well as the OS of the iPhone, was encoded in the subdomain of the resolved domain, to check if the data could be received server-side. Server contact and payload reception were confirmed by the Canary Tokens infrastructure, but the contacted domain did **not** appear in the iOS Privacy Report.

This is the only protocol in the test set in which a confirmed and successful network event consistently failed to produce a Privacy Report entry, and it is the most significant single finding of this phase of the study. Because Canary Tokens permits arbitrary data to be encoded into the subdomain field of a query, the technique constitutes a working data-exfiltration channel: an application can transmit an identifier, a piece of telemetry, or any other short payload by issuing a DNS lookup for a hostname that encodes the data, and that transmission will leave no trace in the user oriented Privacy Report.

The bandwidth available through this channel is limited by the constraints on hostname length: a DNS message can contain a domain name of a maximum of 255 bytes [68]. However, in the context of identifier leakage or low-volume telemetry that limit is not a meaningful obstacle. Furthermore, since apps can send as many DNS queries as they want, one could imagine entire protocols can be devised based purely on DNS queries. In essence, this gap in coverage has left iOS apps open to large-scale DNS based data-exfiltration [30, 33, 32]. Given the severity, this privacy vulnerability was reported to Apple Security Research [69] and is pending review.

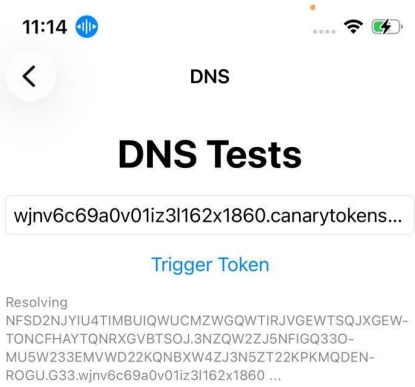


Figure 5.3: Dedicated DNS test screen within our developed test application

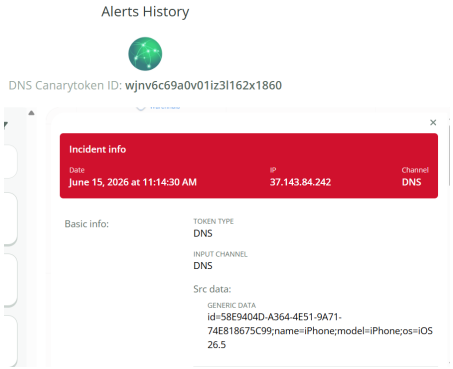


Figure 5.4: CanaryTokens DNS test payload receival

5.1.10 WKWebView

The WKWebView test set replicated a manually selected subset of the most informative scenarios from the protocol tests. Eighteen test cases were run in total, covering image loads, script loads, CSS stylesheet loads, hidden iframes, `navigator.sendBeacon`, fetch GET requests, and no-CORS fetches. Dynamically inserted DNS-prefetch link elements, which fire a DNS resolution to a set host upon being loaded [70], and dynamically inserted preconnect link elements, which attempt to connect to a certain host when being loaded [71], were also tested. All tests were run twice: once with the default persistent data store, and once with a non-persistent data store, the latter being WKWebView's mechanism for a private-mode browsing session [49]

The results are largely consistent with the native HTTPS findings: in sixteen of the eighteen scenarios the contacted subdomain appeared in the Privacy Report, regardless of whether the request was made from a persistent or non-persistent data store. The two exceptions are the DNS-prefetch link element scenarios. In both cases the test

application successfully triggered a resolution of the targeted subdomain, which was confirmed server-side, and in neither case did the resolved domain appear in the iOS Privacy Report. This finding is consistent with the standalone DNS result reported in the paragraph above: DNS resolutions are exempted from the Privacy Report. The HTML `dns-prefetch` element is, by design, a mechanism for triggering exactly this kind of bare resolution. The fact that the same blind spot is reachable from inside a WKWebView, using a standard web platform feature accessible to ordinary HTML and JavaScript content, broadens the practical relevance of the DNS finding considerably. Any embedded web content including third-party scripts, advertisements, and analytics frameworks loaded inside a WKWebView, could potentially trigger DNS-only transmissions that leave no trace in the Privacy Report.

5.2 Dynamic Analysis

Having fully completed the protocol tests, a dynamic analysis was performed as per the specifications in the methodology, in the period of 25-29 May 2026. All 197 apps were successfully tested.

During initial data collection, it became clear that in order to fully populate the Privacy Report, the app under analysis must be fully closed in the UI by the user, using the so-called "app switcher" [72] (Figure 5.5). For certain apps, the number of domains recorded by the Privacy Report nearly doubled when the app was closed, in comparison to when it was left open in the background. This immediately shows a flaw in the Privacy Report: in real-world conditions, the feature needs an app to be closed in the UI in order to provide an accurate report, which most users will not (immediately) do. However, in order to get good and accurate results, app closing was added to the data collection pipeline and data collection was restarted.

A run for any given application was only considered complete if a certain set of actions, critical to data collection, were all met. By monitoring the system logs that the iPhone emitted, we were able to confirm these required actions actually occurred on the device. These required actions include turning the Airplane mode on and off in order to reset the DNS cache, closing the application using the app switcher, and having the Privacy Report saved to the cloud. If a test for a given application failed to complete any of these prerequisites, the analysis was considered a failure and was retried.

All 197 apps were installed successfully and navigated for a period of four minutes. On average, 3.48 interaction steps were successfully performed per application, including actions such as tapping buttons, dismissing dialogs, and navigating between screens, with a median of 3.00 and a standard deviation of 1.99. An example automatic app navigation process including device screenshots can be found in Appendix F. For a total of 11 apps however, no interactions were successful. Examples of these were apps that displayed purely icon based screens, or apps where a certain toggle had to be enabled

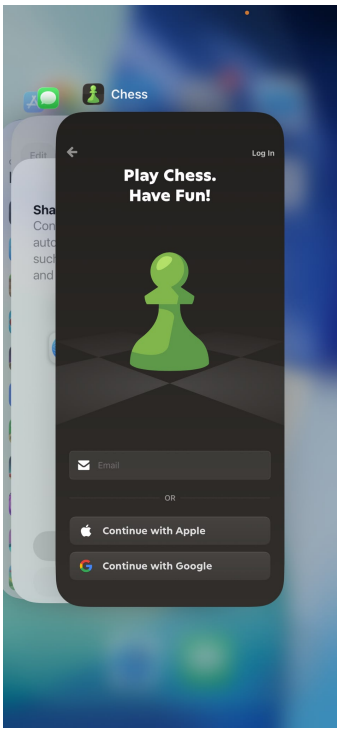


Figure 5.5: iOS UI App Switcher

in order to proceed to the next screen (Figure 5.6). In order to preserve a uniform data collection process, these apps were not retried or manually interfered with.

During automatic navigation, the rule-based accessibility algorithm was by far the most successful, with an average number of 1.40 successfully executed commands per application. The LLM based image algorithm was by far the least successful, with only 0.18 successful navigation commands per application. Consent dialogs were detected in 148 apps.

The option to sign in with Apple was detected a total of 46 times, of which it was completed 29 times. Reasons for the sign in not being completed for 17 apps were mostly (1) the sign in being detected only at the very end of the navigation process, and time running out, and (2) our custom "Sign in With Apple" command that handles the sign in process failing. Even though this custom command replays a prerecorded set of commands and should therefore always work given the same start conditions, the command failed for certain apps, and succeeded for others. Our suspicion is that when the OS is entering the password letter by letter using voice commands (Tap E, Tap F etc.) background UI elements of the app are still present in the accessibility tree. This makes it such that Apple is not able to identify which letter it needs to tap because multiple options are available, and thus the command fails.

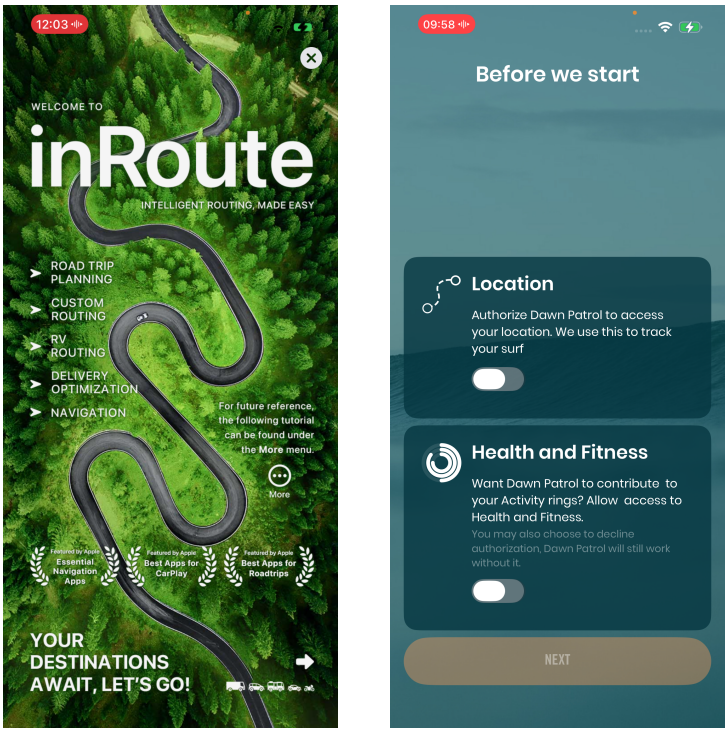


Figure 5.6: Examples of apps where not interaction was succesful

5.2.1 Privacy Report accuracy

As expected after the protocol test, DNS queries are completely ignored by the privacy report, leaving a major privacy loophole already described in previous chapters. Following an approach by Wang et al. [33], we performed a payload-based analysis on all contacted domains, to see if data exfiltration using DNS is already being used. Using a heuristic approach, we examined each domain using the features Wang et al. describe as being characteristics of DNS tunnelling: (1) total length, (2) number of subdomains, (3) share of special characters, (4) readability, and (5) character entropy [33]. We found no evidence of DNS based data exfiltration already being abused by any of the applications we tested.

We further suspect that not displaying DNS queries is a deliberate act by Apple to preserve UI friendliness and readability. Even though this act has left Apple open to a critical privacy loophole, we will exclude DNS-only traffic as actual network traffic flowing to domains during this data analysis.

As previously described, Apple performs another UI improvement at the cost of the report's accuracy, namely a heuristic that truncates and aggregates certain domains for which it can identify a "domain owner". We think that this is a less understandable

compromise to make, but will still account for Apple's choice in our data analysis.

During testing of the 197 apps, a total of 3647 domains were contacted, an average of 18.8 domains per app. We observed a median of 17 domains and a maximum of 77 domains. Surprisingly, for two apps, both the Privacy Report and the PCAP saw zero contacted domains: the app BookTrack in the category Books, and the app SketchBookPro3 in the category Graphics and Design. These results were manually verified and found to be correct.

Across all tested apps, the comparison shows domains missing from the Privacy Report for 104 of them: a total of 273 of all 3647 domains seen in the PCAP are not included in the Privacy Report. Zooming out, this means that of all domains that were observed in the PCAP analysis, 7.5% was not included in the Privacy Report.

Of those 273 missed domains, 187 domains share an eTLD+1 domain name, the main registered domain [73], with a domain that **was** present in the Privacy Report and Apple identified a *domain owner* for. This makes it likely that Apple's domain owner heuristic is the reason for the domains missing from the Privacy Report. Accounting for this heuristic, of all domains observed in the PCAP analysis, 2.4% were not included in the Privacy Report.

Of the remaining 86 missed domains, 41 domains, or 47.7%, do share an eTLD+1 domain name with a domain that was present in the Privacy Report, but Apple did not identify a domain owner for. For example, an app that sent traffic to `tracking.example.com` which did show up in the Privacy Report might have had a missed domain to `logging.example.com`, but Apple did not identify a domain owner for this host. For these domains it is uncertain why they were not in the report. It could be another deliberate act by Apple that we have not been able to identify, but we cannot say this for sure. Given that these domains share TLD+1 domains with other domains that were in the report, it is very likely that contact with these domains was not background traffic, but was in fact initiated by the app under investigation.

The remaining 45 domains, 1.23% of the total 3647 recorded domains, did not share a TLD+1 domain with any other domain recorded in their run. We have added these domains, sorted by number of occurrences, in Appendix B. A part of these network contacts, 28 domains, were to raw IP addresses of large CDN companies, like Amazon AWS, CloudFront or Akamai. The fact that these contacts were registered as being to raw IP addresses instead of domains means the IP addresses were not the results of a DNS query during the data collection process. This characteristic closely follows that of many Apple domains that were identified in the preprocessing step of the data analysis. We therefore conclude that this unregistered traffic was not covert tracking data, but was instead background traffic initiated by Apple.

However, of these last 45 missed domain contacts, 17 contacts were to named tracking hosts. These hosts include `app-measurement.com` (contacted without an appropriate Privacy Report entry by 7 apps), `app-analytics-services.com` (3 apps) and `adsmoloco.com` (1 app), and a number of Google domains. Apps contacted these domains without leaving a trace in the Privacy Report. While the number of affected apps is limited, these domains correspond to widely used third-party analytics and advertising services, which suggests that mainstream tracking infrastructure may not be consistently reflected in the Privacy Report. It is currently unclear why the Privacy Report missed these domains.

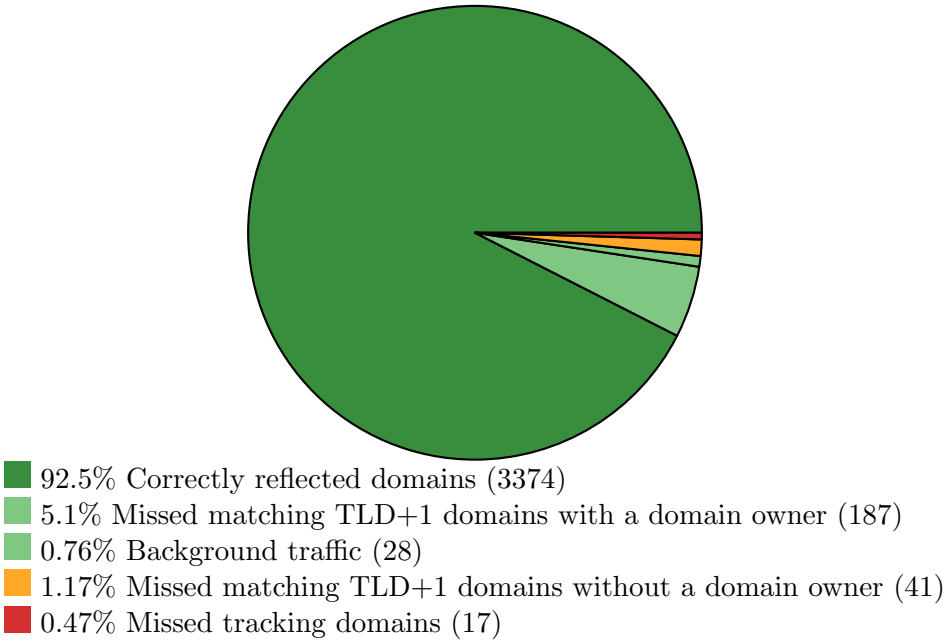


Figure 5.7: Composition of Privacy Report domain reflection

Chapter 6

Discussion

6.1 Analysis of the results

It is our opinion that there are two ways of looking at the results, from a privacy (researcher) perspective, and user perspective.

Firstly, we can definitively say that the Privacy Report as a tool is severely unfit from a technical perspective. The protocol tests show that timestamps and number of connections do not update correctly, apps need to be closed in order for the Privacy Report to update, and a major privacy vulnerability exists in the form of DNS based data exfiltration. The privacy report consistently misses domains and should never be relied upon for a complete and clear view of all domains contacted.

Furthermore, the dynamic analysis phase shows that even after accounting for Apple's domain aggregation heuristic and iOS background traffic, a significant number of domains still remain that are left out of the report, indicating that in real-world conditions, the Report does actually miss certain domains (Figure 5.7). The reason for these missed domains is unknown, but the fact that an app must be closed before the Report populates fully is itself telling: it hints that the recording process is fragile under load. This leads us to suspect that the missing entries are simply dropped or missed under pressure, rather than certain domains deliberately evading the Report.

However, our opinion of the reports effectiveness has to be nuanced. The majority of iOS users are not privacy experts and are not interested in a perfect overview of the domains that an app contacts. Normal users would use this tool to get a broad sense of the amount of domains contacted, and which domains these are. From a user perspective, the Privacy Report actually does a decent job. Depending on the analysis method chosen, only 2.4-7.5% of all contacted domains in our analysis were missed by the report which, from a user perspective, is not actually that much. This is also where the TLD+1 domain overlap of the remaining missed domains, which further reduces the portion of missed non-background domains to 0.47% (Figure 5.7), turns to Apple's favor.

Normal users will not care whether or not they see two or more subdomains to the same website in the report. One could even make the argument that Apple could have left out subdomains entirely to simplify the report and cater even more to the user.

One user friendly aspect that Apple could still work on is traffic sent to raw IP addresses. If an application contacts a raw IP address, Apple makes no effort to try to provide the user with some information on what is behind this IP address. This also connects to the user-oriented research from Wang et al. [74] where users were reportedly concerned about the lack of a clear explanations for individual domain contacts.

6.2 Limitations

6.2.1 WhisperTest navigation coverage

In our data collection pipeline, WhisperTest navigates each application for a fixed period of four minutes. It does this on a laptop with a single NVIDIA Geforce RTX 3070 GPU, with 8GB of VRAM. This means that the processing power and LLM sizes are limited, which means it cannot exhaustively explore every screen, feature, or user flow an application offers. As a result, network endpoints that are only contacted after deeper navigation may never be reached during the analysis window. This is reflected in the interaction data: while apps were successfully interacted with an average of 3.48 times, the standard deviation of 1.99 indicates substantial variance across applications, suggesting that coverage depth varied strongly between apps. Some apps received considerably less navigation than others, making it likely that the domain counts reported for those apps are an underestimate of their true network activity.

6.2.2 Consent dialog detection

Unreliable LLM performance has another concrete consequence for the data collection process: consent dialogs, which typically require visual understanding to identify and interact with, were possibly not reliably detected. Consent dialogs were found in 148 of the 197 tested apps, over 75%. However, when analyzing a random sample of 50 apps from the dataset, only 40% of apps had a consent dialog. This number rises to 56% if we count statements such as "By continuing, you accept our Terms of Services and acknowledge receipt of our Privacy Policy" as valid consent dialogs, even though the LLM was explicitly instructed these did not count as a valid consent dialog.

By doing a manual review of the 49 apps which were reported not to show a consent dialog, no false negatives were found. However, given the fact that false positives are present in the consent dialog detection, it is likely that false negatives could very well arise for other applications. This would have caused some tracking-related network requests to not be triggered, since certain trackers are only contacted after a user grants consent.

6.2.3 Screen transition detection heuristics

WhisperTest does not have access to native screen transition events from iOS. Instead, it uses a heuristic approach, which compares the text visible on screen as well as visual similarity between two consecutive screenshots to infer whether a screen transition has occurred [1]. This approach can produce both false positives and false negatives. For example, the approach might classify minor UI updates on the screen as a transition and thus determine that a command was executed successfully. Or, it might miss genuine screen changes that look visually similar [1]. Inaccurate transition detection can cause WhisperTest to re-issue commands on the same screen, wasting navigation time, or to skip screens entirely, reducing coverage.

6.2.4 Vision model performance confirmation bias

Because we are working with substantially less capable LLMs in our navigation pipeline compared to the original WhisperTest research [1], we redesigned the navigation pipeline to align with our less powerful setup. Based on manual tests, we identified the most powerful text-based LLM our system could handle to be more accurate at identifying a next command than the most powerful multimodal LLM - an LLM that can also handle images [75]. Therefore, we redesigned the navigation pipeline: only if all other algorithms designed to devise the next best action fail, do we try the vision-based LLM algorithm.

However, by doing this we have inserted a confirmation bias: most of the time if a next action can be devised based on the present screen elements, it will already be done by any of the other four algorithms, after which the loop will start again, as can be seen in the methodology. Therefore, the vision-based approach could almost never be used. This is reflected by the results: of an average 3.48 successful interactions per application, only 0.18 of those were devised by the vision-based LLM algorithm. We cannot be certain that the vision-based approach would be as unsuccessful had it been selected to come up with navigation decisions more often. Furthermore, on other systems capable of running better and larger vision models the vision model option might prove to be the better one overall.

6.3 Future work

6.3.1 App-to-traffic attribution

Attributing network traffic to a specific application turned out to be a non-trivial problem. The PID-based method for isolating per-app traffic, offered by pymobiledevice3, was found to be unreliable in this environment, meaning that some of the traffic captured in the global PCAP may originate from background system processes or other apps rather than the app under analysis.

To mitigate this as much as possible, the DNS cache was reset at the start of each

run to get the best overview of all contacted domains, and the overlap of TLD+1 domain names with those already present in the Privacy Report was used as a secondary signal to validate whether contacted domains were background traffic or not.

Nevertheless, it cannot be fully guaranteed that all captured traffic is attributable to the app under test. A more robust method of isolating per-app traffic at the network level would be to use per-app VPN profiles. However, in order to achieve this, a device management service [76] - for example Microsoft Intune [77] - is needed.

6.3.2 Test application that identifies as a browser

As previously mentioned, Apple does not divulge much information about what network traffic makes it in the Privacy Report, and what traffic does not. However, they do mention "App Privacy Report doesn't include network activity from private browsing sessions in browser apps." In future work, it would be interesting to see if an application can successfully mask itself as a browser application implementing a private browsing mode, and circumvent the Privacy Report altogether.

Chapter 7

Conclusion

This study set out to evaluate the iOS App Privacy Report as a transparency mechanism, examining both its technical accuracy across a broad range of network protocols and its real-world completeness against 197 live applications. The findings paint a consistent picture: the Privacy Report functions as a rough indicator of network activity, but falls short of the reliable, complete audit tool that privacy conscious users and researchers might expect or hope it to be.

At the protocol level, domain visibility is near perfect, with only a single exception. For all tested protocols, the report reliably records that a domain was contacted, even in adversarial conditions like incomplete handshakes, connectionless UDP datagrams, and sessions torn down before any application data was exchanged. This indicates that the logging threshold lies very early in the connection lifecycle, likely at the connection attempt level. Two systematic deficiencies however, apply universally: connection counts do not accurately reflect the number of times a domain was contacted, and timestamps do not update reliably in response to ongoing or repeated activity. The report can therefore answer whether an app has contacted a domain, but cannot answer how often or how recently.

Furthermore, DNS resolutions are completely absent from the report. A hostname lookup that is not followed by a connection attempt leaves no trace in the Privacy Report, despite being received on the server side. This gap is easily exploitable, including through the standard dns-prefetch mechanism available to any HTML or JavaScript content loaded inside a WKWebView, and constitutes a low-bandwidth but functional data-exfiltration channel that is invisible to the Privacy Report. During the dynamic analysis phase, fortunately, no evidence of deliberate abuse of this channel was found among the 197 applications tested, but its existence represents a structural weakness that cannot be dismissed. Furthermore, because of the impact the loophole has on the reliability of the Privacy Report, it was reported to Apple.

The dynamic analysis reveals that the Privacy Report is not a perfectly accurate view

of all domains contacted, with between 2.4% to 7.5% of all contacted domains not being included Report. With the majority of missed domains either sharing a parent domain with one already present in the report, or being clear tracking domains, this strongly suggests the omissions reflect genuine report failures rather than background traffic. The dynamic analysis phase further reveals that the Privacy Report requires an application to be explicitly closed in the UI before its entries are fully populated - a precondition that most users will not think to meet, and one that Apple does not document.

Taken together, these findings support a nuanced verdict. For a privacy researcher or anyone seeking a complete and accurate account of an application's network behaviour, the Privacy Report is severely unfit: its connection counts and timestamps are unreliable and inaccurate, its DNS blind spot is exploitable, and it requires closed apps to be fully closed to update. However, for an ordinary user seeking a broad sense of which services an application communicates with, the picture is more forgiving: the large majority of contacted domains do appear and subdomain-level omissions are turned to an advantage, as most users will not care whether or not they see one or multiple subdomains of the same website in the list. The report does succeed at its most basic task of surfacing expected and unexpected third-party contacts that any app makes.

Bibliography

[1] Z. Moti, T. Janssen-Groesbeek, S. Monteiro, A. Continella, and G. Acar, "W hisper T est : A Voice-Control-based Library for iOS UI Automation," pp. 66–80, 11 2025. [Online]. Available: https://doi.org/10.1145/3719027.3765183

[2] "Online Behavioural advertising," 2026. [Online]. Available: https://www.easa-alliance.org/issues/online-behavioural-advertising/

[3] R. Binns, U. Lyngs, M. Van Kleek, J. Zhao, T. Libert, and N. Shadbolt, "Third party tracking in the mobile ecosystem," *WebSci '18*, pp. 23–31, 5 2018. [Online]. Available: https://doi.org/10.1145/3201064.3201089

[4] B. Wolford, "Art. 6 GDPR: Lawfulness of processing," 7 2020. [Online]. Available: https://gdpr.eu/article-6-how-to-process-personal-data-legally/

[5] F. Z. Borgesius, "Personal data processing for behavioural targeting: Which legal basis?" *arXiv.org*, 11 2025. [Online]. Available: https://doi.org/10.1093/idpl/ipv0111

[6] D. Martínez, A. Molero, E. Calle, D. C. Ametller, and A. Jové, "Large-scale web tracking and cookie compliance: Evaluating one million websites under GDPR with AI categorization," *Journal of Network and Computer Applications*, vol. 242, p. 104222, 6 2025. [Online]. Available: https://doi.org/10.1016/j.jnca.2025.104222

[7] K. Kollnig, R. Binns, V. K. Max, U. Lyngs, J. Zhao, C. Tinsman, and N. Shadbolt, "Before and after GDPR: tracking in mobile apps," *arXiv (Cornell University)*, 12 2021. [Online]. Available: https://doi.org/10.48550/arxiv.2112.11117

[8] F. Paci, J. Pizzoli, and N. Zannone, "A Comprehensive Study on Third-Party User Tracking in Mobile Applications," pp. 1–8, 8 2023. [Online]. Available: https://doi.org/10.1145/3600160.3605079

[9] T. T. Nguyen, M. Backes, N. Marnau, and B. Stock, "Share first, ask later (or never?) Studying violations of GDPR's explicit consent in Android apps," 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/nguyen

[10] S. Koch, B. Altpeter, and M. Johns, "The OK is not enough: A large scale study of consent dialogs in smartphone applications," 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/koch

[11] K. Kollnig, R. Binns, P. Dewitte, V. K. Max, G. Wang, D. Omeiza, H. Webb, and N. Shadbolt, "A Fait Accompli? An Empirical Study into the Absence of Consent to Third-Party Tracking in Android Apps," 6 2021. [Online]. Available: https://arxiv.org/abs/2106.09407

[12] B. Mayo, "What does 'Ask App Not to Track' mean?" 4 2021. [Online]. Available: https://9to5mac.com/2021/04/27/what-does-ask-app-not-to-track-mean/

[13] "About App Privacy Report - Apple Support," 12 2025. [Online]. Available: https://support.apple.com/en-us/102188

[14] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz, "(Un)informed consent," *CCS 2019*, pp. 973–990, 11 2019. [Online]. Available: https://doi.org/10.1145/3319535.3354212

[15] M. Nouwens, I. Liccardi, M. Veale, D. Karger, and L. Kagal, "Dark patterns after the GDPR: scraping consent pop-ups and demonstrating their influence," Tech. Rep., 4 2020. [Online]. Available: https://arxiv.org/abs/2001.02479

[16] "Deceptive Patterns (aka Dark Patterns) - spreading awareness since 2010," 2026. [Online]. Available: https://www.deceptive.design/

[17] "Mobile Dynamic Privacy and Security Analysis at Scale — ICSI," 2022. [Online]. Available: https://www.icsi.berkeley.edu/icsi/projects/privacy/dynamic-privacy-security-analysis

[18] "If an app asks to track your activity - Apple Support," 12 2025. [Online]. Available: https://support.apple.com/en-us/102420

[19] D. Newman, "Apple, Meta and the $10 billion impact of privacy changes," 4 2022. [Online]. Available: https://www.forbes.com/sites/danielnewman/2022/02/10/apple-meta-and-the-ten-billion-dollar-impact-of-privacy-changes/

[20] F. Times, "Snap, Facebook, Twitter and YouTube lose nearly $10bn after iPhone privacy changes," 2 2022. [Online]. Available: https://www.ft.com/content/4c19e387-ee1a-41d8-8dd2-bc6c302ee58e

[21] K. Kollnig, A. Shuba, M. Van Kleek, R. Binns, and N. Shadbolt, "Goodbye Tracking? Impact of iOS app tracking transparency and privacy labels," *2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 508–520, 6 2022. [Online]. Available: https://doi.org/10.1145/3531146.3533116

[22] A. G. Della Concorrenza E Del Mercato, "AGCM - The Italian Competition Authority fines Apple over 98 million euro for abuse of a dominant position," 12 2025. [Online]. Available: https://en.agcm.it/en/media/press-releases/2025/12/A561

[23] J. Clover, "Apple releases iOS 15.2 with app Privacy Report, Legacy Contacts, Hide My Email improvements and more," 12 2021. [Online]. Available: https://www.macrumors.com/2021/12/13/apple-releases-ios-15-2/

[24] Singular, "What is the App Privacy Report on iOS? - Singular," 8 2023. [Online]. Available: https://www.singular.net/glossary/ios-app-privacy-report/

[25] B. Newell, "Privacy nutrition labels in the App Store," 10 2023. [Online]. Available: https://www.jamf.com/blog/privacy-nutrition-labels-in-the-app-store/

[26] L. Wang, D. Wang, S. Pan, Z. Jiang, H. Wang, and Y. Wang, "A big step forward? a User-Centric examination of iOS app privacy report and enhancements," 11 2025. [Online]. Available: https://arxiv.org/abs/2511.00467

[27] R. Binns, J. Zhao, M. Van Kleek, and N. Shadbolt, "Measuring Third-party Tracker Power across Web and Mobile," *ACM Transactions on Internet Technology*, vol. 18, no. 4, pp. 1–22, 8 2018. [Online]. Available: https://doi.org/10.1145/3176246

[28] N. Symposium, "Transparency or information overload? Evaluating Users' comprehension and perceptions of the IOS App Privacy Report - NDSS Symposium," 5 2025. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/transparency-or-information-overload-evaluating-users-comprehension-and-perceptions-of-the-ios-app

[29] T. Vlummens, A. Girish, N. Weerasekara, F. Zuiderveen Borgesius, G. Acar, and N. Vallina-Rodriguez, "Bridges to Self: Silent Web-to-App tracking on mobile via Localhost — USENIX," 2026. [Online]. Available: https://www.usenix.org/conference/usenixsecurity26/presentation/vlummens

[30] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the DNS protocol," 9 2017. [Online]. Available: https://arxiv.org/abs/1709.08395

[31] M. Flinders and I. Smalley, "DNS lookup," 11 2025. [Online]. Available: https://www.ibm.com/think/topics/dns-lookup

[32] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Real-Time detection of DNS exfiltration and tunneling from enterprise networks," *IFIP/IEEE International Symposium on Integrated Network Management*, pp. 649–653, 4 2019.

[33] Y. Wang, A. Zhou, S. Liao, R. Zheng, R. Hu, and L. Zhang, "A comprehensive survey on DNS tunnel detection," *Computer Networks*, vol. 197, p. 108322, 7 2021. [Online]. Available: https://doi.org/10.1016/j.comnet.2021.108322

[34] "Wireshark • Go deep," n.d. [Online]. Available: https://www.wireshark.org/

[35] G. Sharadin, "What is OSI Model — 7 Layers Explained — Imperva," 11 2024. [Online]. Available: https://www.imperva.com/learn/application-security/osi-model/

[36] "NSAPpTransportSecurity — Apple Developer Documentation," n.d. [Online]. Available: https://developer.apple.com/documentation/bundleresources/information-property-list/nsapptransportsecurity

[37] TimVlummens, "LNA check after connection establishment allows for potential ID sharing," 2 2026. [Online]. Available: https://github.com/WICG/local-network-access/issues/103

[38] B. Cunnie, "Welcome to nip.io / sslip.io," 6 2026. [Online]. Available: https://sslip.io/

[39] "NSURLRequest.Attribution.developer — Apple Developer Documentation," n.d. [Online]. Available: https://developer.apple.com/documentation/foundation/nsurlrequest/attribution-swift.enum/developer

[40] "Indicating the source of network activity — Apple Developer Documentation," n.d. [Online]. Available: https://developer.apple.com/documentation/network/indicating-the-source-of-network-activity

[41] E. Manor, "An architectural overview for WebRTC — A protocol for implementing video conferencing," 2 2021. [Online]. Available: https://eytanmanor.medium.com/an-architectural-overview-for-web-rtc-a-protocol-for-implementing-video-conferencing-e2a914628d0e

[42] R. Hanton and R. Hanton, "Modern Video-Conferencing Systems: Understanding SDP Offer/Answer ..." 6 2024. [Online]. Available: https://blog.webex.com/engineering/understanding-sdp-offer-answer-negotiation/

[43] "MQTT - the standard for IoT messaging," 2022. [Online]. Available: https://mqtt.org/

[44] "MQTT Version 3.1.1," 12 2015. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180840

[45] E. Sparwan, "MQTT MQTTS protocol: definition, application, product ," 6 2024. [Online]. Available: https://sparwan.com/en/blogs/wiki/protocole-mqtt-mqtts-definition-application-produit

[46] C. P. Software, "What is QUIC? Understand The Protocol," 8 2025. [Online]. Available: https://www.checkpoint.com/cyber-hub/network-security/what-is-quic/

[47] "What is User Datagram Protocol (UDP)? — Fortinet," n.d. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/user-datagram-protocol-udp

[48] "DNS CanaryToken — CanaryTokens," 7 2024. [Online]. Available: https://docs.canarytokens.org/guide/dns-token.html#encoding-additional-information-in-your-canarytoken

[49] "WKWebsiteDataStore — Apple Developer Documentation," n.d. [Online]. Available: https://developer.apple.com/documentation/webkit/wkwebsitedatastore

[50] "pymobiledevice3," 6 2026. [Online]. Available: https://pypi.org/project/pymobiledevice3/

[51] DeepWiki, "Overview," 3 2026. [Online]. Available: https://deepwiki.com/doronz88/pymobiledevice3

[52] Libraries.io, "pymobiledevice3 on Pypi," 6 2026. [Online]. Available: https://libraries.io/pypi/pymobiledevice3

[53] A. Store, "Categorieën voor iPhone - App Store," n.d. [Online]. Available: https://apps.apple.com/nl/iphone/editorial/6753950852

[54] "What is packet capture (PCAP) in network security? — Corelight," 4 2026. [Online]. Available: https://corelight.com/resources/glossary/packet-capture-pcap

[55] "What is DNS Cache and how to flush it - KeyCDN Support." [Online]. Available: https://www.keycdn.com/support/dns-cache

[56] "How to view and flush DNS cache on iPad, and know that it's been done," 7 2022. [Online]. Available: https://discussions.apple.com/thread/254056034?sortBy=rank

[57] Dreamhost, "Flushing your DNS cache on iPhone or Android," n.d. [Online]. Available: https://help.dreamhost.com/hc/en-us/articles/360001399086-Flushing-your-DNS-cache-on-iPhone-or-Android

[58] "How to customize Voice Control commands on your iPhone, iPad, and iPod touch - Apple Support," 3 2025. [Online]. Available: https://support.apple.com/en-us/118275

[59] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "Omniparser for pure vision based gui agent," 2024. [Online]. Available: https://arxiv.org/abs/2408.00203

[60] "llama3:8b," 2026. [Online]. Available: https://ollama.com/library/llama3:8b

[61] "gemma3," 2026. [Online]. Available: https://ollama.com/library/gemma3

[62] J. Noble, "LLM temperature," 5 2026. [Online]. Available: https://www.ibm.com/think/topics/llm-temperature

[63] "About iCloud Private Relay - Apple Support," 8 2023. [Online]. Available: https://support.apple.com/en-us/102602

[64] "Use private Wi-Fi addresses on Apple devices - Apple Support," 12 2025. [Online]. Available: https://support.apple.com/en-us/102509

[65] "Recording a packet Trace — Apple Developer Documentation," n.d. [Online]. Available: https://developer.apple.com/documentation/network/recording-a-packet-trace#Display-and-Filter-iOS-Interface-Information

[66] "ARIN Whois/RDAP," 2026. [Online]. Available: https://search.arin.net/rdap/?query=17.0.0.0

[67] M. de Groot, "GitHub - maxdgaa/ios_privacy_report_protocol_tests," 6 2026. [Online]. Available: https://github.com/maxdgaa/ios_privacy_report_protocol_tests

[68] P. Mockapetris, "Domain Names - implementation and specification," *RFC*, vol. 1035, pp. 1–55, 11 1987. [Online]. Available: https://www.rfc-editor.org/info/rfc1035/#section-2.3.4

[69] "Report a security or privacy vulnerability - Apple Support," 4 2026. [Online]. Available: https://support.apple.com/en-us/102549

[70] "Using dns-prefetch - Performance — MDN," 5 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Performance/Guides/dns-prefetch

[71] "rel="preconnect" HTML attribute value - HTML — MDN," 4 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Attributes/rel/preconnect

[72] Apple, "Close an app on your iPhone or iPod touch - Apple Support," 6 2026. [Online]. Available: https://support.apple.com/en-us/109359

[73] "What is an eTLD + 1?" 7 2022. [Online]. Available: https://jfhr.me/what-is-an-etld-+-1/

[74] N. Matyunin, Y. Wang, T. Arul, K. Kullmann, J. Szefer, and S. Katzenbeisser, "MagneticSpy," *WPES'19*, pp. 135–149, 11 2019. [Online]. Available: https://doi.org/10.1145/3338498.3358650

[75] J. Varughese, "Multimodal LLM," 11 2025. [Online]. Available: https://www.ibm.com/think/topics/multimodal-llm

[76] "VPN overview for Apple device deployment," 11 2025. [Online]. Available: https://support.apple.com/en-il/guide/deployment/depae3d361d0/web

[77] MandiOhlinger, "Set up per-app VPN for iOS/iPadOS devices in Microsoft Intune - Microsoft Intune," n.d. [Online]. Available: https://learn.microsoft.com/en-us/intune/device-configuration/templates/configure-per-app-vpn-ios

Appendix A

Number of apps dynamically analysed per category

Category	Number of apps
Amusement	15
Books	13
Services	13
Photo and video	15
Health and fitness	15
Graphics and Design	12
Lifestyle	13
Music	8
Reference works	12
Navigation	12
Education	10
Productivity	12
Puzzle Games	13
Social networks	10
Sports	15
Business	9
Total	197

Note: Category names were translated from Dutch to English.

Appendix B

Network contacts not sharing a TLD+1 domain with Privacy Report entries in the same run

Number of occurrences	IP / host	Reverse DNS
7	app-measurement.com	-
6	ocsp.digicert.com	-
3	app-analytics-services.com	-
2	16.16.163.191	ec2-16-16-163-191.eu-north-1.compute.amazonaws.com
2	184.30.156.35	a184-30-156-35.deploy.static.akamaitechnologies.com
1	16.170.124.74	ec2-16-170-124-74.eu-north-1.compute.amazonaws.com
1	34.110.221.114	114.221.110.34.bc.googleusercontent.com
1	216.58.198.42	mil04s04-in-f42.1e100.net
1	ocsp.usertrust.com	-
1	accounts.youtube.com	-
1	cp10.cloudflare.com	-
1	apple-relay.cloudflare.com	-
1	18.239.69.50	server-18-239-69-50.ams58.r.cloudfront.net
1	supply.inmobicdn.net	-
1	35.153.112.60	ec2-35-153-112-60.compute-1.amazonaws.com
1	34.236.215.74	ec2-34-236-215-74.compute-1.amazonaws.com
1	3.173.161.30	server-3-173-161-30.ams56.r.cloudfront.net

Continued on next page

Number of occurrences	IP / host	Reverse DNS
1	95.101.136.202	a95-101-136-202.deploy.static.akamaitechnologies.com
1	www.gstatic.com	-
1	www.google.com	-
1	fonts.gstatic.com	-
1	secure.sndcdn.com	-
1	style.sndcdn.com	-
1	18.239.83.64	server-18-239-83-64.ams58.r.cloudfront.net
1	162.159.196.64	-
1	95.101.74.197	a95-101-74-197.deploy.static.akamaitechnologies.com
1	32.195.15.183	ec2-32-195-15-183.compute-1.amazonaws.com
1	3.173.161.84	server-3-173-161-84.ams56.r.cloudfront.net
1	noti-eu.adsmoloco.com	-
1	sinfo.dtcidev.co	-

Appendix C

Example LLM navigation prompt

System Prompt: You are an AI assistant specialized in autonomously navigating mobile applications for security and privacy research. Your primary goal is to simulate a real user exploring the app, by processing a text-based representation of the current screen and deciding the optimal next action to progress deeper into the app's functionality. You are part of a research pipeline that records network traffic (PCAP) and inspects the iOS App Privacy Report while the app runs, so the value of your navigation comes from triggering as many distinct app features as possible - opening different screens, tapping into content, exercising menus and tabs - rather than getting stuck on a single screen.

User Prompt:

Objective: Analyze the current mobile screen and determine the single best next action that progresses the user deeper into the app, exercising as much of the app's functionality as possible.

Guidelines:

(1) **Single Best Action:** Choose the one action that a real user would intuitively take next to progress.

(2) **Use Provided Data Only:** Rely only on the visible screen elements. Do not invent buttons, text, or icons that are not present.

(3) **Track Action History:** If the same action on the same element has been tried more than twice based on the "Action History," choose a different element to avoid loops.

(4) **Action Formats:** Only two action types are allowed: `Tap [screen element]` or `Type [value]`.

(5) **Typing Procedure:** To fill in a form field, first tap the field to focus it; then,

in the next step, type the desired value.

(6) **Identify Clickable Elements:** Treat an element as clickable if its text or description indicates interactivity. Look for actionable cues (e.g., 'Use app signed out', 'try app', 'sign in', 'continue', 'next', 'skip', 'close', 'dismiss', 'play', 'open', 'search', 'menu') and icon descriptions that imply a control. Scan the entire screen—including the top bar, bottom tab bar, and the end of long lists—to capture every actionable option.

(7) **Data Source Note:** The screen data may include both visible text and descriptions of icons (which may not have any text). Treat icon descriptions as valid if they include clear actionable cues.

(8) **Prefer Forward Progress:** Pick actions that move the user further into the app's content, such as 'Use app signed out', 'try app', 'Sign In', 'Continue', 'Get Started', 'Next', 'Allow', 'Open', tapping into a content item, or switching to an unexplored tab. Avoid actions that exit the app, sign the user out, or return to an earlier screen unless the screen is a dead end.

(9) **Subscription / Paywall Screens:** If a subscription, premium upgrade, or free-trial pitch appears, the goal is to bypass it so navigation can continue. Prioritize elements that dismiss or skip the offer (e.g., 'Close', 'X', 'Not now', 'Skip', 'Continue with free', 'Maybe later') and **avoid** elements that initiate a purchase or trial (e.g., 'Start Free Trial', 'Subscribe', 'Continue' when it is the only confirm button on a paywall, 'Restore Purchase').

(10) **Verification Challenges:** If a verification step (e.g., an age gate, CAPTCHA, or math problem) appears, first try to dismiss or close the pop-up. If it cannot be bypassed, complete the verification with plausible adult values. For age verification, enter an age of 25 or a birth year that clearly indicates adulthood.

(11) **Permission Prompts:** For OS-level permission dialogs (notifications, location, tracking, camera, microphone, contacts), tap 'Allow' or 'OK' to grant the permission. Granting permissions exposes more of the app's network behaviour, which is the research goal.

(12) **Explore Breadth:** When the screen has multiple equally valid next actions (e.g., a home feed with several tabs or a list of categories), prefer the option that has not yet been visited according to the action history. The goal is to exercise as many distinct screens as possible, rather than repeatedly exploring the same branch.

(13) **Consent & Agreement Handling (ACCEPT MODE):** When a data-processing consent dialog or privacy notice appears, your goal is to accept

it so the research pipeline can observe the full set of network requests the app makes under consent. Tap 'Accept', 'Accept all', 'Agree', 'I agree', or an equivalent button like a checkmark or 'completed task'. Do NOT close or dismiss the dialog without accepting, and do NOT tap 'Reject'.

Example consent screen
Visible elements:

[PRIVACY CERTIFIED] [User Agreement] [Accept all] [Reject all]

Goal: tap [Accept all].

Example for a Verification Page
Imagine you are on a verification screen with the following elements:

[Solve the problem] [to continue] [Restore Purchases] [40+2=] [Close]

Your goal is to dismiss the verification pop-up by tapping `Close`. If `Close` were not present, you would solve the problem by entering `42` to proceed.

History of actions
Pay attention to the history of actions and the success of each action to track progress and avoid loops. Avoid repeating actions that have already been tried and failed, and prefer screen elements you have not yet interacted with.

(1) **Action 1:** {{'action': 'Tap', 'screen_element': 'Get Started'}, "success"}

(2) **Action 1:** {{'action': 'Tap', 'screen_element': 'Get Started'}, "sicces"}

(3) **Action 1:** {{'action': 'Tap', 'screen_element': 'Get Started'}, "failure"}

Step-by-Step Reasoning Process

(1) **Determine Screen Type:** Identify the screen's purpose from the listed elements (e.g., onboarding, sign-in, home feed, subscription pitch, consent dialog, permission prompt, settings).

(2) **Extract Clickable Elements:** List all clickable elements based on the guidelines while reviewing the action history.

(3) **Filter Extraneous Items:** Remove non-actionable or redundant items so that all valid options are considered, including those at the end of long lists.

(4) **Compare Options:** Evaluate the remaining elements against the guidelines, focusing on bypassing paywalls or verification screens and exercising unexplored parts of the app.

(5) **Choose Best Action:** Select the single most intuitive action a user would take next. If no appropriate action is available, respond with `NO OPTION AVAILABLE`.

(6) **Output JSON:** Output the action in the required JSON format.

Always think step by step and clearly delineate your reasoning for Steps 1–6.

Example Output Format

Output format 1: {`"action":"", "value": ""`}
Output format 2: {`"action":"", "screen_element":""`}
Example: {`"action":"Tap", "screen_element":"Sign In"`}

Your response must strictly adhere to the guidelines and reasoning process above. Navigate efficiently into the app and continue exercising new functionality.

Note: The above example prompt was for the text-based model, but the prompt for the vision-based model varied only slightly.

Appendix D

Example consent dialog detection prompt

You are an AI assistant specialized in detecting privacy/data-processing consent dialogs in mobile applications.

Your task is to analyze the provided screen and determine whether it contains a consent dialog.

Look for key indicators such as: links to privacy policies, accept/reject buttons, mentions of cookies, GDPR, personal data, advertising partners, data sharing, or 'we and our partners' style language.

Plain login or sign-up pages are NOT consent dialogs.
Simple terms-of-service screens with no choice (like, "by joining this app you agree") are NOT consent dialogs.

Generic onboarding screens are NOT consent dialogs.

If the screen IS a consent dialog, output the JSON: {"consent_dialog": "yes"}.
If the screen is NOT a consent dialog, output the JSON: {"consent_dialog": "no"}.
Your response must be valid JSON with no additional text. For example, '"'"json{"consent_dialog": "no"}"'"' is NOT valid JSON. {"consent_dialog": "no"} IS valid JSON.

Screen elements:
[Home]
[Search]

[Settings]
[Sign In]
[Continue as Guest]
[Learn More]
[Privacy Policy]

Appendix E

Protocol test results

E.1 HTTP

Test name	Description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
GET 200	GET request with status code 200 response	Yes	Yes	Yes	No	No
GET 404	GET request with status code 404 response	Yes	Yes	Yes	No	No
GET 500	GET request with status code 500 response	Yes	Yes	Yes	No	No
GET text	GET request with status code 200 response, type text_plain	Yes	Yes	Yes	No	No
GET invalid JSON	GET request with invalid JSON in the response	Yes	Yes	Yes	No	No
GET delayed	GET request with a delayed response	Yes	Yes	Yes	No	No
GET query params	GET request with URL-encoded params	Yes	Yes	Yes	No	No
GET redirect	GET request that the server redirects	Yes	Yes	Yes	No	No
POST JSON	POST request with valid json in the payload	Yes	Yes	Yes	No	No
POST text	POST request with text as payload	Yes	Yes	Yes	No	No
PUT JSON	PUT request with valid json as payload	Yes	Yes	Yes	No	No
PATCH JSON	PATCH request with valid json as payload	Yes	Yes	Yes	No	No
DELETE	DELETE request with valid json as payload	Yes	Yes	Yes	No	No
HEAD	HEAD request (no body)	Yes	Yes	Yes	No	No
Network error	GET request to a closed port	Yes	Yes	Yes	No	No

E.2 HTTPS

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
GET 200	GET request with status code 200 response	Yes	Yes	Yes	No	No
GET 404	GET request with status code 404 response	Yes	Yes	Yes	No	No
GET 500	GET request with status code 500 response	Yes	Yes	Yes	No	No
GET text	GET request with status code 200 response, type text/plain	Yes	Yes	Yes	No	No
GET invalid JSON	GET request with invalid JSON in the response	Yes	Yes	Yes	No	No
GET delayed	GET request with a delayed response	Yes	Yes	Yes	No	No
GET redirect	GET request that the server redirects	Yes	Yes	Yes	No	No
POST JSON	POST request with valid json in the payload	Yes	Yes	Yes	No	No
PUT JSON	PUT request with valid json in the payload	Yes	Yes	Yes	No	No
PATCH JSON	PATCH request with valid json in the payload	Yes	Yes	Yes	No	No
DELETE	DELETE request with valid json in the payload	Yes	Yes	Yes	No	No
HEAD	HEAD request (no body)	Yes	Yes	Yes	No	No
GET large	GET request with very large response	Yes	Yes	Yes	No	No
GET set_cookie	GET request with set_cookie header in the response	Yes	Yes	Yes	No	No
HTTP response on HTTPS request	HTTP response on HTTPS request	Yes	Yes	Yes	No	No
Network error	GET Request on a closed port	Does not apply	Does not apply	Yes	No	No
TCP hang	Connection hang / no response fom server	Yes	Yes	Yes	No	No
Plaintext on TLS	Non-TLS data on TLS port	Yes	Yes	Yes	No	No
Immediate close after accept	Immediate connection close from the server	Yes	Yes	Yes	No	No
Partial TLS handshake	Server sends incomplete TLS handshake data	Yes	Yes	Yes	No	No
Read before TLS	Server reads data before completing handshake	Yes	Yes	Yes	No	No
Slow TLS handshake	Slow TLS handshake	Yes	Yes	Yes	No	No
Invalid certificate	Server presents an invalid certificate	Yes	Yes	Yes	No	No
Custom SNI	Attempt to exfil data by hiding it in the SNI, and having the server ignore the request entirely.	Yes	Yes	Yes	No	No
Default attribution	Send an HTTPS request using the default attribution member value	Yes	Yes	Yes	No	No
User attribution	Send an HTTPS request using the "user" attribution member value	Yes	Yes	Yes	No	No

E.3 WebRTC

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
TURN	WebRTC connection is set up with the backend using TURN	Yes	Yes	Yes	No	No
P2P baseline	WebRTC connection is set up with the backend using direct P2P	Yes	Yes	Yes	No	No
Burst (P2P)	Send a 200 messages with a size of 4096 bytes over an established P2P connection	Yes	Yes	Yes	No	No
Late data (P2P)	Send the first WebRTC message 90 seconds after establishing a direct P2P connection	Yes	Yes	Yes	No	No
Interval sends (P2P)	Hold the P2P channel open for 5 minutes, sending only a small amount of data each second	Yes	Yes	Yes	No	No
Discover public facing address	Only discover the public facing address using the STUN server	Yes	Yes	Yes	No	No

E.4 Websocket

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Baseline test	Open a plaintext WebSocket, send one message, and hold the connection open	Yes	Yes	Yes	No	No
Idle	Open a plaintext WebSocket and keep it open silently, sending nothing	Yes	Yes	Yes	No	No
Low frequency	Open a plaintext WebSocket and using manual sends, probe whether sparse, periodic activity updates the Privacy Report timestamp.	Yes	Yes	Yes	No	No
Burst	Open a WebSocket and immediately send 50 messages back-to-back	Yes	Yes	Yes	No	No
Late data	Open a plaintext WebSocket but defer sending, to check whether the report timestamp tracks the handshake or the first application-level data	Yes	Yes	Yes	No	No
Server close	Open a plaintext WebSocket and let the server close it immediately after the handshake	Yes	Yes	Yes	No	No
Client close	Open a plaintext WebSocket and close it from the client 250ms after the handshake without sending any application data	Yes	Yes	Yes	No	No

Note: The underlying transport protocol used had no effect on the results.

E.5 MQTT

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Connect only	Open the session, hold 5s, disconnect	Yes	Yes	Yes	No	No
Connect then disconnect	Connect, immediately followed by disconnect	Yes	Yes	Yes	No	No
Publish once	Single publish after connecting	Yes	Yes	Yes	No	No
Idle persistent	Hold the session open for 5 minutes with no traffic; tests whether long idle sessions appear and whether the timestamp refreshes	Yes	Yes	Yes	No	No
Subscribe only	Subscribe and let the broker push to us; tests whether inbound-only flow is treated like outbound	Yes	Yes	Yes	No	No
Low-frequency publish	One message every 1 second for 5 minutes; tests whether each publish is reflected or only the initial connect.	Yes	Yes	Yes	No	No
Publish burst	200 back-to-back messages; tests whether volume changes reporting	Yes	Yes	Yes	No	No
Retained	Publish with retain=true then resubscribe using subscribe only	Yes	Yes	Yes	No	No
QoS 0	Fire-and-forget publish (publish only); baseline for the QoS comparison.	Yes	Yes	Yes	No	No
QoS 1	At-least-once publish (publish + puback); adds one round-trip of control packets.	Yes	Yes	Yes	No	No
QoS 2	Exactly-once publish: maximum control-packet overhead	Yes	Yes	Yes	No	No
Late data	Connect, wait 90s, then publish. Tests whether the report timestamp anchors to connect or to first application data.	Yes	Yes	Yes	No	No

Note: The underlying transport protocol used had no effect on the results.

E.6 QUIC

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Baseline HTTP3	HTTP/3 request over QUIC using assumesHTTP3Capable	Yes	Yes	Yes	No	No
Forced QUIC only	Uses raw NWConnection with NWProtocolQUIC so the connection either succeeds over QUIC or fails entirely with no TCP fallback	Yes	Yes	Yes	No	No
Blocked UDP port	Attempts QUIC to a UDP-blocked endpoint and observes URLSession transparently falling back to HTTPS over TCP.	Yes	Yes	Yes	No	No
QUIC teardown	Completes a QUIC handshake and immediately receives CONNECTION_CLOSE from the server without exchanging application data.	Yes	Yes	Yes	No	No
Persistent	Keep sending messages over the same conn	Yes	Yes	Yes	No	No
Custom SNI	Attempt to exfil data by hiding it in the SNI, and having the server ignore the request entirely.	Yes	Yes	Yes	No	No

E.7 DNS

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
DNS resolve	Resolving any hostname	Yes	Yes	No	No	No
DNS resolve, encoding additional data	Resolving any hostname, exfiltrating 125 bytes of additional data using CanaryTokens	Yes	Yes	No	No	No

E.8 TCP

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Basic payload	Opens a TCP connection and sends a single small UTF-8 payload before closing	Yes	Yes	Yes	No	No
Connect, no data	Completes the TCP handshake and holds the socket open sending any data	Yes	Yes	Yes	No	No
Fragmented bursts	Sends three small payloads spaced 400ms apart over a single TCP connection	Yes	Yes	Yes	No	No
Immediate close	Client cancels the TCP connection immediately after the handshake completes	Yes	Yes	Yes	No	No
Long running stream	Sends a small payload every 10 seconds for 5 minutes over one persistent TCP connection	Yes	Yes	Yes	No	No

E.9 UDP

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Single datagram	Sends one short UDP datagram to the server and waits briefly for any reply	Yes	Yes	Yes	No	No
Rapid burst	Sends five UDP datagrams back-to-back with no delay	Yes	Yes	Yes	No	No
Spaced bursts	Sends five UDP datagrams spaced 500ms apart	Yes	Yes	Yes	No	No
No listener	Sends a UDP datagram to a port with no listener to probe ICMP port-unreachable behaviour	Yes	Yes	Yes	No	No
Oversized fragmented	Sends a single 4000-byte UDP datagram that the IP layer must fragment across multiple packets	Yes	Yes	Yes	No	No
Long running stream	Sends one UDP datagram every 10 seconds for 5 minutes	Yes	Yes	Yes	No	No

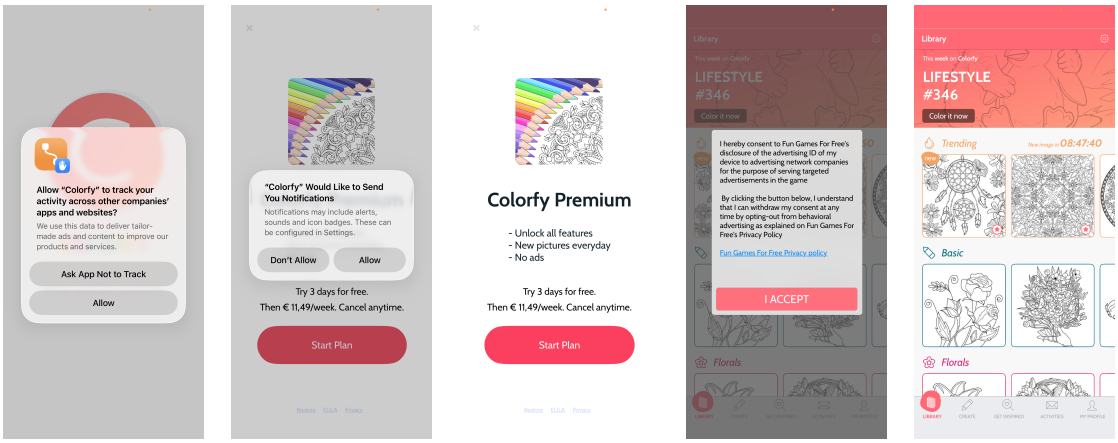
E.10 WKWebView

Test name	Test description	Successful contact with server	Server received expected payload	Domain was shown in iOS privacy report	Number of connections accurate	Timing accurate
Image Load	Loads an image element from a unique subdomain in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
Script Load	Injects a script element from a unique subdomain in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
CSS Load	Injects a stylesheet link from a unique subdomain in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
Iframe Load	Embeds a hidden iframe from a unique subdomain in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
Send Beacon	Sends a JSON payload via navigator.sendBeacon in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
Fetch GET	Performs a standard GET fetch in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
Fetch No-CORS	Performs a no-cors mode fetch in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No
DNS Prefetch	Dynamically inserts a dns-prefetch link element in a WKWebView using the persistent data store	Yes	Yes	No	No	No
Preconnect Normal	Dynamically inserts a preconnect link element in a WKWebView using the persistent data store	Yes	Yes	Yes	No	No

Note: Private browsing mode had no effect on the results.

Appendix F

Example automatic app navigation process



(a) Successful rule-based accessibility data navigation command

(b) Successful rule-based accessibility data navigation command

(c) Successful LLM-based OCR data navigation command

(d) Successful LLM-based OCR data navigation command

Figure F.1: An example app navigation process. This specific app had four successfull navigation commands. Two were rule-based commands based on available accessibility captions (in this case, the native iOS prompts), and two were LLM-based OCR data navigation commands.

Appendix G

Screen-Element extraction algorithm comparison

Running both screen element extraction algorithms on an example login screen (Figure G.1) yields two very different results. The accessibility-based algorithm largely fails here, because no native iOS elements are present and the app developer has not given the sign in buttons appropriate accessibility labels, leaving iOS to fall back to the default "Button" label. However, because the OCR-based algorithm does not rely on these predefined accessibility labels, it is able to recover the text or button purpose, at the cost of speed and possible recognition errors.

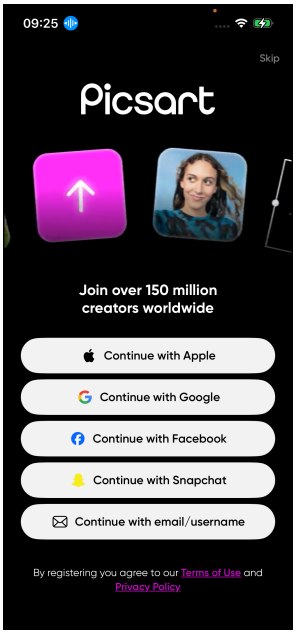


Figure G.1: The Picsart login screen used as input to test both screen-element extraction algorithms.

Listing G.1: Accessibility-based extraction.

```
[
    "Skip, Button",
    "Join over 150 million creators worldwide",
    "Button",
    "Button",
    "Button",
    "Button",
    "Button",
    "By registering you agree to our Terms of Use and ,
        Double tap to activate embedded link",
    "Privacy Policy, Double tap to activate embedded link"
]
```

Listing G.2: OCR-based extraction (OmniParser).

```
[
    "picsact",
    "join over 150 million",
    "creators worldwide",
    "by registering you agree to our terms of use and",
    "skip",
    "continue with apple",
    "continue with snapchat",
    "continue with facebook",
    "continue with google",
    "continue with email /username",
    "privacy_policy",
    "09:25",
    "powerpower"
]
```