Analyzing the iOS Local Network Permission from a Technical and User Perspective

David Schmidt*†, Alexander Ponticello¶§, Magdalena Steinböck*, Katharina Krombholz¶, Martina Lindorfer*
*TU Wien †CDL AsTra ¶CISPA Helmholtz Center for Information Security §Saarland University
*{david.schmidt, magdalena.steinboeck, martina}@seclab.wien ¶{alexander.ponticello, krombholz}@cispa.de

***Abstract*—In the past, malicious apps attacked routers or identified locations through local network communication. To mitigate security and privacy risks from local network access, Apple introduced a new permission with iOS 14. To be effective, the permission needs to protect against technical threats, and users must be able to make an informed permission decision. The latter is presumably hindered by the intrinsic technicality of the concept of the local network.**

In this paper, we perform the first comprehensive analysis of the local network permission by studying four key aspects. We investigate the security of its implementation by systematically accessing the local network. We explore local network accesses via a large-scale dynamic analysis of 10,862 iOS and Android apps. We analyze the concepts that constitute the permission prompts, as this is all the information users get before making a decision. Based on the identified concepts, we conduct an online survey ($N = 150$) to comprehend users' understanding of the permission, their threat awareness, and common misconceptions.

Our work reveals two methods to bypass the permission from webviews, and that the protected local network addresses are insufficient. We show how and when apps access the local network, and how the situation differs between iOS and Android. Finally, we present the light and shadow of users' understanding of the permission. While nearly every participant is aware of at least one threat (83.11%), misconceptions are even more common (84.46%).

1. Introduction

With the release of iOS 14 in 2020, Apple introduced a new permission to guard access to the local network [16]. Usually, devices connected to a local network cannot be directly accessed from the Internet. However, if apps can access a local network, they can communicate with these devices, opening new attack vectors. Furthermore, collecting information of a person's local network can enable tracking and user profiling [48], [78]. After the permission became widely used, a discussion arose among users as to why certain apps requested such access. Popular apps like AliExpress and Instagram were suspected of scanning the network and collecting data to track users [79], [83].

Prior research already identified threats induced by local network access. Reardon et al. [78] uncovered Android apps that obtained the MAC address of WiFi routers through local network communication to bypass the location permission. Girish et al. [48] demonstrated that household fingerprinting and user tracking are possible via local multicasts. Furthermore, a malicious Android app was found reconfiguring the Domain Name System (DNS) settings of routers to forward traffic to malicious domains [58]. Finally, Kuchhal and Li [60] explored local network access of websites with desktop browsers and observed legitimate use cases such as fraud and bot detection, but warned of the potential misuse of this technique for tracking and profiling.

Prior research has not yet investigated the situation when a permission protects local network access nor the iOS permission itself. Noteworthy, this permission is distinct from others: (1) It affects not only the device running the app but also its surroundings, i.e., all devices connected to the network. (2) Android, as of version 15 (released in 2024), does not have a corresponding permission that restricts accessing the local network. (3) Compared to other permissions, like location or contact access, it is presumably less obvious to users what the local network comprises and why information about it is sensitive. (4) Developers cannot directly request the permission from the app's code. Instead, the Operating System (OS) detects access and prompts the user to grant the permission [20].

The implementation of the local network permissions is also yet to be studied. Users raised concerns that YouTube was bypassing the local network permission [81]. In this case, the access was caused by Apple's AirPlay functionality, which does not require permission. However, unauthorized access might be possible. Further, it is unclear how prevalent local network access is and if apps behave differently on Android, where no such permission exists.

To fill these gaps, we address this topic with an interdisciplinary approach by looking at both the technical and user perspective. We contribute a comprehensive assessment of the security and privacy of this functionality, how apps access the local network, how they prompt users for it, and the users' understanding of the permission. It is vital to study both perspectives. The technical aspects give us insights into potential misuse of the permission and how widespread local network access is in apps. The user perspective helps us understand whether the permission dialog enables users to make informed decisions. Furthermore, it shows threat awareness related to local network access. Both technical and user aspects need to be effective, as none can achieve sufficient protection without the other. If the permission is

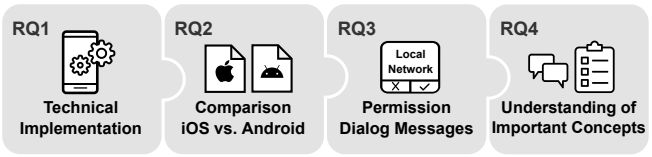

Figure 1: Overview of analysis to answer our research questions. (RQ1) We study in Section 3 if the local network permission is implemented securely. (RQ2) In Section 4, we show how prevalent local network access is. (RQ3) Section 5 presents our content analysis of the permission rationales. (RQ4) To investigate the users' perspective, we performed a user study as described in Section 6.

not implemented securely, there is a false sense of security and privacy. Likewise, if users are not able to make an informed decision, malicious apps could trick users into granting the permission. Thus, as shown in the overview in Figure 1, we investigate the technical and user perspective in four consecutive analysis steps.

First, we investigate whether the implementation of the permission is secure or if it is possible to bypass it. We systematically access the local network to find potential gaps in its enforcement to answer *RQ1: Is the local network permission implemented securely?* Next, we inspect how widespread local network access is in apps and study how the situation differs between iOS and Android by performing a large-scale analysis of apps available on both platforms. We dynamically analyze 10,862 apps present on both platforms to answer *RQ2: How prevalent is local network access in apps?* We further investigate the information of the permission prompts, where developers can provide a rationale explaining why access is requested. We extract these messages and analyze their contents to answer *RQ3: Which concepts constitute the permission prompts?* Finally, we perform an online user study with 150 participants to shed light on the users' understanding of the local network permission and their awareness of threats in this context to ultimately answer *RQ4: What is the user's understanding of these concepts?*

In summary, we make the following contributions:

- We demonstrate two methods to bypass the permission and show that the protected local IP address range of the permission is insufficient.
- We analyze 10,862 cross-platform apps, out of which 152 iOS and 117 Android apps access the local network, and show differences between both platforms.
- We identify reoccurring concepts in permission prompts and present insights into the developer-specified purposes.
- We show that nearly every participant (83.11%) is aware of at least one threat but also that misconceptions are widespread, as 84.46% hold at least one.

Artifacts. For reproducibility and to enable future work, we publish our code to study the permission, analyze apps, extract and label permission messages, and evaluate the results at: https://github.com/SecPriv/local_network.

2. Background and Motivation

In this section, we give an overview of what we consider as a local network, provide threats posed by apps that can access the local network, and discuss the permission on iOS protecting the local network access.

2.1. Local Network

We consider the WiFi network to which a mobile phone is connected to as the local network. This is not restricted to users' home network but can also be, e.g., a company network or the WiFi of a café.

Accessible Devices. Connected devices, like the phone itself, have an IP address assigned within the subnet mask of the network. However, depending on the network setup, devices with local IP addresses outside of the subnet mask may also be reachable. Hence, threats that affect devices within the subnet mask also apply to locally reachable devices outside the subnet mask. Therefore, we consider them as part of the local network.

Virtual Private Networks (VPNs). In Section 3, we also consider VPNs connected to the phone as similar to a local network. Technically, VPNs are not a local network, but the security and privacy threats we consider also apply to devices within VPNs. Devices within local networks and VPNs are typically not reachable from the Internet. However, they are still accessible from other devices within the network. For VPNs, threats can even have worse consequences, as companies use VPNs for internal services.

2.2. Threat Model

Security. Devices connected to a local network are usually not directly accessible from the Internet as a firewall protects them. However, they are still accessible on the local network. Thus, malicious apps running on a user's phone can try to attack other devices on the local network [58], [94]. This is in particular a concern in smart homes, as Internet of Things (IoT) devices are often configured with default passwords and rarely updated, a fact heavily exploited by IoT malware [2], [4].

Location. Local network access also brings threats to the user's privacy. Apps can use the router's MAC address to look up location information in online databases [67], [88]. In the past, apps used that to bypass the location permission [77], [78]. If an app observes the devices in the user's local network, it can infer location changes and behavior patterns since local networks differ in terms of used IP addresses and connected devices.

Advertisements. Knowledge about connected devices can be relevant for advertisements (ads) agencies. They could decide which ads to show based on the devices on the local network. In online discussions, users raised concerns that AliExpress might use local network information for this purpose [83]. Also, users might not want to share which devices

they own with every app, e.g., health-related devices with insurance apps. We have seen that websites use information about users' devices to show different prices, e.g., charging Apple users more, so-called personal pricing [47]. With local network access, other devices connected to the local network might also influence the prices, making it harder to bypass. Cross-device and cross-user tracking is another privacy threat, as local network information gathered on different devices connected to the same network can be used to link users [48]. Device names of connected devices can be sensitive as well, e.g., if users use their real name [59].

Misconceptions. In a congress hearing, the TikTok CEO was asked if TikTok accesses other devices on the WiFi network. Online users argued that the questioner did not understand how WiFi networks work as the app needs it to access the Internet. This shows that there is a lack of understanding of the local network with its security and privacy implications [54], [93]. If this misconception is widespread, malicious apps could exploit it to trick users into granting permission. For example, apps could tell their users that they require local network access to download large files via WiFi, and if they do not grant it, it could lead to additional costs when downloaded via cellular Internet.

2.3. Permission Overview

Terminology. Android restricts apps from accessing specific data or performing actions through permissions. There exist two major types of permissions: (1) Install-time permissions, granted by the system upon app installation if apps declare them [9], and (2) runtime permissions, also called dangerous permissions, which require user confirmation [9].

There are similar concepts on iOS. Apps can declare *entitlements* to gain additional capabilities, which iOS grants upon installation [25]. However, if apps access so-called protected resources, they require user consent [29]. For handling the consent request and keeping track of the user decisions, the Transparency, Consent, and Control (TCC) framework is responsible [35], [52]. In the following, we use the term permissions for actions that require user consent. Otherwise, we state it explicitly.

Local Network Permission on iOS. According to Apple, all outgoing traffic to a local network, multicast, and broadcast addresses requires the local network permission [19]. Technically, iOS does the permission check at the lowest system level, and thus, it should include all network APIs. Apple mentions that they plan to extend the permission to incoming multicast and broadcast traffic [19], but have not yet implemented it as of iOS 18 (released in 2024).

An exception to the permission are Bonjour services provided by the OS for AirPlay and printing. Bonjour is a zero-configuration protocol that allows the discovery of devices and services on the local network. Apple argues that those methods do not reveal any network details to the apps; Thus, they do not require the permission [19]. AirPlay is integrated into different parts of iOS to offer functionalities to stream audio or video to other devices on the local network, e.g., the iOS media player and webviews send such requests when playing a video [14]. However, this does not trigger the permission, as the local network information is not provided to the app.

Compared to other iOS permissions, the process to request the local network access permission differs, which could influence the user's understanding. App developers cannot request permission directly. Instead, the OS asks the user to grant permission if an app tries to access the local network for the first time [30]. All further attempts to access the local network do not trigger the permission prompt. Users can only change their decision in the settings or have to reinstall the app [18].

To help users understand why apps require the local network permission, app developers can add descriptions to the permission request. If developers do not provide a so-called permission rationale (also named permission purpose string), iOS shows the default message "*This app will be able to discover and connect to devices on the networks you use.*" along a description always provided by the system "*[App name] would like to find and connect to devices on your local network*."

Lack of Permission on Android. Android, as of version 15 (released in 2024), does not have a comparable local network access permission. While the `CHANGE_WIFI_MULTICAST_STATE` [8] permission exists that allows apps to receive multicast messages, Android ranked its protection level as `"normal"`; thus, it is an install-time permission and does not require user confirmation [9]. Moreover, apps can still send messages to the local network without holding this permission, as it only restricts apps from receiving multicast messages [11].

3. Permission Implementation

First, we test when iOS enforces the local network permission to gain further insights into its implementation and answer *RQ1: Is the local network permission implemented securely?* To do so, we systematically try to access different addresses through various methods.

3.1. Methodology

To gain insights when iOS enforces the local network permission, we developed a test app. The app tries to contact different broadcast, local network, and multicast addresses. Additionally, we varied the methods to access the network to see if the permission is enforced for all network APIs [19].

We installed the test app on an iPhone 8 running iOS 16 (released in 2022) jailbroken with palera1n [71]. We used tcpdump [100] on the phone to capture its traffic. Moreover, we connected the phone to a VPN to test the protection of the VPN address space for which the same security threats apply. To ensure the jailbreak did not influence our findings and that they still exist in later versions, we retested them manually on an iPhone SE 2020 that was not jailbroken running iOS 17.3 (released in 2024).

We tried to cover as many methods to perform local network requests as possible. Therefore, we consulted Apple's developer documentation [24]. In total, we identified 16 different methods that allowed us to perform ICMP, TCP, UDP, and QUIC requests. In addition to the explicitly mentioned protocols in the Frequently Asked Questions (FAQ) [19] (TCP and UDP), we included ICMP to test a different Internet layer and QUIC for another transport layer protocol. For each method, we attempted to contact different addresses: local addresses within and outside the connected WiFi network mask, IPv4 and IPv6 multicast addresses, IPv4 broadcast addresses, and local IPv6 addresses. In Section A.1, we provide an overview of tested addresses, and in Table 1 the methods we used.

We included local IPv6 addresses in our tests despite the address space of 2^{128}. On the one hand, scanning the whole address space by contacting each address is not practical. On the other hand, if an app learns the addresses of connected devices, they are well suited for user tracking, as different networks likely use different combinations of addresses. IPv6 could be especially suited for tracking if hosts generate their addresses using stateless autoconfiguration based on MAC addresses [101]. On local IPv6 networks, multicasts and neighbor discovery exist to find devices. However, we could not test sending neighbor discovery messages, as iOS restricts app access to raw sockets [21].

To identify the tested method in the traffic dump, we performed a GET request to a remote server under our control, containing a test ID in the URL path before and after each test case. We executed our test app twice, once granting permission to ensure the app sends the request and once denying it to observe if it is enforced.

3.2. Results

Based on our systematic tests to trigger the local network permission, we not only found two methods that bypass it (Section 3.2.1), but also that the protected address space is insufficient (Section 3.2.2), as well as a lack of rationales (Section 3.2.3). Table 1 summarizes our results.

3.2.1. Permission Bypass. We discovered that iOS does not correctly enforce the local network permission for two methods that allow including web content in apps (so-called webviews). Compared to Android webviews, for which security and privacy aspects have been well studied [36], [37], [68], iOS offers different types of webview components with different levels of embedding within the host app.

SFSafariViewController. The SFSafariViewController [32] is part of the SafariServices framework and is similar to a Safari window within the app. Apple advises using it to visit external websites [17]. The opened website cannot directly send data back to the app or execute code from the app. However, the website loaded within those components could scan the local network, retrieve privacy-sensitive data by communicating with connected devices or attack connected devices [1], [53]. Thus, the same threat model also applies to websites and their running code [60].

TABLE 1: Results of tested methods. ✖ indicates that the app sent the request without the permission, thus bypassing it. ━ indicates that Apple intended that the functionality did not require permission even if it accessed the local network. ✔ indicates that iOS correctly enforced the permission. No symbol means that sending such a request was impossible. *Local* means the local address was within the connected WiFi's subnet, and *local outside* outside of it.

Protocol	Methods	Local	Local outside	Multi-cast	Broad-cast
ICMP	SimplePing [15]	✔	✖		
QUIC	NWConnection	✔	✖	✔	✔
TCP	NSURLConnection	✔	✖		
	NWConnection	✔	✖		
	SendTo	✔	✖		
	SFSafariViewController	✖	✖		
	UIWebview	✔	✖		
	URLSession	✔	✖		
	WKWebView	✖	✖		
UDP	AirPlay			━	
	AirPrint			━	
	NetServiceBrowser			✔	
	NSBonjourServices			✔	
	NWConnection	✔	✖	✔	✔
	SendTo	✔	✖	✔	✔
	ServiceBrowser			✔	

WKWebView. WebViews allow embedding web content directly into an app and can interact with the app [17]. In iOS 8 (released in 2014), Apple added the WKWebView [34], to replace the older UIWebView [33]. The UIWebView has been deprecated since iOS 12 (released in 2016). We found that iOS correctly enforces the permission for the deprecated UIWebView, in contrast to the recommended WKWebView. With WKWebViews, apps can load their own JavaScript code from the app and send the retrieved information back to the app. For example, they could try to load web content from devices connected to the local network; in the case of the Philips Hue bridge, this reveals its device type, and, if the path `/description.xml` is also accessed, its model and serial number. The behavior directly contradicts Apple's developer FAQ, which states that the permission is required for all outgoing traffic, including WKWebView [19].

Case Study: Browsers. Since Apple required third-party web browsers to build upon WebKit [73], they also used WKWebView to render web content. Thus, they can bypass the permission. With Google Chrome [50], Opera [69], Microsoft Edge [64], and Brave [39], it is possible to observe the WKWebView permission bypass. To load the web content in the webview, they send a GET request to `/favicon.ico` to show an icon in the header bar if a user visits a webpage within the local network. iOS loads the page before the user allows it, since the web content loaded with the WKWebView bypasses the permission. The icon request is done outside the WKWebView and triggers the permission dialogue. Other browsers like Safari [31] and

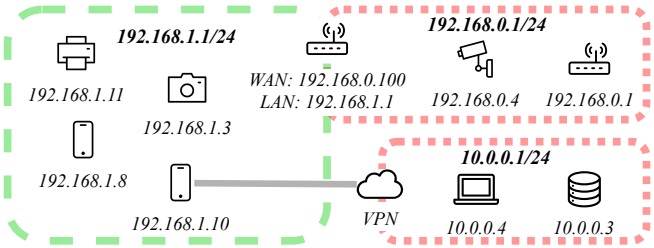

Figure 2: Vulnerable local network example. The router of the local network `192.168.1.1/24` forwards packets to the `192.168.0.1/24` network. The iPhone with the address `192.168.1.10` is also connected to a VPN. We colored the network protected by the permission green (dashed) and the addresses not protected red (dotted).

Firefox [66] bypass the permission too. However, as they do not perform a request outside the webview, they do not trigger the permission dialogue.

3.2.2. Insufficient Protection. The permission is also insufficient for complex networks and VPNs. We observed that apps can send messages to local addresses outside the connected WiFi's subnet mask without the permission.

Figure 2 shows an example of a vulnerable network: The iPhone running the app connects to the WiFi network `192.168.1.1/24`. The router providing access to the iPhone is also part of a different local network, `192.168.0.1/24`, and forwards traffic from `192.168.1.1/24`. Since the permission only protects the `192.168.1.1/24` address space, apps can access devices on `192.168.0.1/24`, for which the same threats apply. Additionally, if the iPhone connects to a VPN, the address space of the VPN is also not protected by the permission. Even if a VPN is technically not a local network, the same security and privacy threat apply: Devices not reachable from the Internet become accessible.

In general, the permission protects access to addresses within the subnet mask of the connected WiFi. That means the protected addresses might not be enough in a complex network setup. In Section 4.3.2, we show that 23 iOS apps tried to access local IP addresses outside the connected network, highlighting the relevance of this finding.

3.2.3. Local Network Rationales. According to Apple's developer documentation, apps must provide a rationale string (also called usage description) to explain why they want to access a protected resource, and if they do not have it, iOS blocks the app's access [29]. We did not observe this behavior regarding the local network, even if it is a protected resource [27]. Apps can acquire local network permission without a rationale. In the documentation of the local network rationales, Apple weakens the requirement by writing that rationales *should* be included when an app accesses the local network. That might be due to the difference in how to acquire the permission [27]. Other permissions restrict apps from calling sensitive methods. In contrast, for the local network permission, there is no sensitive method per se, but the OS detects if an app tries to contact the local network.

Bonjour methods are the most similar local network related methods to other permission-protected methods. We tested these with our test app, as they rely on multicast messages. Apps need to specify Bonjour services in the `Info.plist` with the key `NSBonjourServices` to use those methods [28]. Bonjour service strings can be seen as the name of the service to discover. Surprisingly, iOS also does not enforce a local network rationale string for them.

3.2.4. Responsilbe Disclosure. After our disclosure, Apple ensured us that they are working on fixing the bypass via web components and the insufficient protection in complex networks (reported in June 2022), but we have not been updated about a solution (October 2024). For VPNs (reported in April 2024), Apple did not share our point of view and categorized it as *expected behavior* in August 2024.

Takeaways

To answer RQ1: *Is the local network permission implemented securely?*, we showed:

- The iOS local network permission is not implemented securely. Apps can bypass it using SFSafariViewController and WKWebView.
- The protected address space is insufficient for complex local networks and VPNs.

4. Permission Prevalence

After understanding the local network permission, we study the current situation of local network access of apps. Therefore, we dynamically analyze 10,862 cross-platform apps each on iOS and Android to answer RQ2: *How prevalent is local network access in apps?* In addition to iOS apps, we analyze the same apps on Android to compare their local network access behavior, which we suspect to differ due to the lack of comparable permission on Android.

4.1. Dataset

To study the prevalence of the local network permission in apps, we analyze iOS and Android apps. For better comparability, we focused on apps that are available on both platforms, so-called cross-platform apps, by using the approach from Steinböck et al. [95], which identifies matching app pairs within iOS and Android datasets by comparing metadata like app names, and app descriptions. Additionally, they compare to the matches from Google's migration API, which we used to find further app pairs.

For iOS, we tried downloading the 10,000 most popular apps based on user ratings and 10,000 random apps, resulting in 19,847 successfully downloaded iOS apps. Our initial dataset for Android consists of 19,333 apps, where we attempted to download the 9,629 most popular apps based on the number of downloads (at least 10 million

installations) and 10,000 randomly selected apps. To find out what apps exist, we used the Androzoo dataset [3]. We downloaded both popular and randomly chosen apps to get a holistic representation of what users are confronted with in their daily lives, as well as potential edge cases. We could not download all apps because of geographical blocking by developers or device constraints [61].

Using the above-mentioned matching approach [95], we identified 11,405 cross-platform app pairs: 3,047 apps were already within our sample datasets, further 8,358 matching Android apps the migration API identified and we successfully downloaded 7,815 of them. Hence, our final dataset consists of 10,862 app pairs.

4.2. Methodology

We decided to use dynamic analysis to study the local network access of apps. Compared to other permissions, it is not enough to analyze if specific methods are used, as there is no predefined list of methods that require this permission (see Section 2.3). Apps could use various network functionalities to communicate with the local network, which can be found in nearly every app. Also, there is a wide range of possibilities to obtain or use local network addresses, e.g., if apps want to obfuscate it, they can use numerical values. Since we aim to compare the behavior of iOS and Android apps, and the static detection of local network access is challenging, we opted for an approach that works on both platforms. By dynamically executing apps and observing their network traffic, we can detect access for iOS and Android apps using the same methodology. This approach is similar to how iOS triggers the permission prompt, i.e., by checking the destination of outgoing network traffic. For iOS apps, we further statically extracted the local network permission rationales and the Bonjour strings required for Bonjour multicast methods to study their usage. Additionally, we used the extracted rationales in Section 5 to analyze the concepts provided by developers.

For the dynamic analysis, we set up a test infrastructure to run experiments and automatically interact with apps while capturing traffic to detect local network accesses.

4.2.1. Test Infrastructure. Our test network consisted of a router (Asus AC750), four IoT devices, mobile phones, and a Mac mini (M1, 2020) running macOS 12.5. We connected four IoT devices (Google Chromecast [49], Amazon Alexa speaker [5], Philips Hue [72], and Ikea Trådfri [56]) to observe if apps try to interact with them. We used the Mac mini to manage the interaction with the apps.

To capture the traffic and detect local network access, we ran tcpdump [100] on the phones. We jailbroke four iPhones running iOS 16 (released in 2022) using palera1n [71] and rooted three Pixel 6a running Android 13 (released in 2022) with Magisk [63]. If we found any local network access, we performed a second test round to ensure it originated from the app and that it was no background traffic from the OS. We only considered apps accessing the local network if they did so consistently.

4.2.2. Automatic App Interaction. Previous work [41], [78] used Android monkey [6] for dynamic testing to randomly interact with an app. However, as discussed extensively by related work [40], [85], [86], [106], this might not be sufficient and may not cover relevant code. Therefore, we use a systematic interaction approach that clicks through the User Interface (UI) in a Depth-first search (DFS) manner.

We first ran each app for 30 seconds without interaction, after which we performed 25 interactions to obtain more context of the local network access. An app accessing the local network after an interaction might need it to provide functionality, e.g., to send a command to an IoT device or find other players in a game. Accessing the local network on startup is more suspicious. According to Android's developer documentation [10], it is a bad privacy practice to immediately collect privacy-sensitive data, even if apps potentially require the information later on.

We implemented our app interaction in Python and used Appium [13] to locate and interact with elements on the phone's screen. We automatically performed 25 interaction steps with each app; a step is one action, e.g., a button pressed. We chose 25 steps as a tradeoff between execution time and coverage. The interactions followed a DFS exploration strategy. The interactor saved information about what it had already visited and the interactions it performed. If an interaction led to a new state, the interactor continued there. If it had tried all available interactions, it attempted to go back to the previous state. A state consisted of the currently available UI elements.

We accepted all iOS permissions the app asked for, not only in the interaction phase but also during the non-interaction phase, as we cannot observe local network access without consenting to the iOS permission. We did not consider the interaction with system dialogues as an interaction step, as it is not a direct interaction with the app. This is in line with our approach on Android, for which we granted all permissions using `adb` [7] before starting an app.

4.2.3. Rationales and Bonjour Strings. Developers can add their custom permission rationale messages with the key `NSLocalNetworkUsageDescription` to the `Info.plist` and the localization files called `InfoPlist.strings`. Also, apps declare the Bonjour service strings required for Bonjour methods in the property list file (see Section 3.2.3). We developed a Python script using the plistlib library [75] and regular expressions to extract these values from the property list. In Section 4.3.1, we use it to study their occurrences. In Section 5, we further analyze the concepts used in the permission rationales.

4.3. Results

We evaluate different aspects of local network access. We split them up based on the type of access: (1) broadcast, (2) directly contacting a local address, and (3) multicast. We analyze when the access happens, directly after the start or after user interaction, and show what local network addresses outside the connected WiFi apps try to contact.

4.3.1. Local Network Accesses. We found local network accesses in 152 (1.4%) iOS and 117 (1.08%) Android apps. In Table 2, we provide an overview of accesses.

Access Types. The reason why we found more iOS apps accessing the local network is because of multicast messages. We found them in 117 iOS and 73 Android apps. In contrast, we found more Android apps contacting broadcast and local addresses. We encountered 44 iOS and 53 Android apps using broadcasts, while 18 iOS and 24 Android apps directly contacted a local address. A reason why we found more iOS apps using multicast could be the support for Bonjour multicasts. Evaluating the multicast types based on the addresses, we found the multicast DNS (mDNS) address `224.0.0.251` that Bonjour uses in 98 (83.76% of apps using multicasts) iOS apps, while we found it in 50 (68.49%) Android apps. The multicast address we observed the most for Android apps is `239.255.255.250`, which is related to Simple Service Discovery Protocol (SSDP). We found it in 38 (32.48%) iOS and 56 (76.71%) Android apps.

We consider AirPlay as local network access of the OS and not the app. As mentioned in Section 2.3, it does not trigger a permission request as no sensitive data is accessible by the app. Further, we found that playing audio or video triggered iOS to send AirPlay mDNS requests in the background. We also observed similar behavior in Android, which sent mDNS requests to find Google cast devices. Therefore, we do not treat this as app traffic but OS related. Consequently, we did not include it in the local network accesses. Overall, we observed this behavior in 401 (3.69%) iOS and 308 (2.84%) Android apps.

On iOS, we found 14 apps that triggered the permission, but we could not observe any local network accesses in the traffic. We manually analyzed them and found that two apps tried to contact their own WiFi IP address. In that case, the OS does not forward it to the WiFi interface but only to the loopback interface. Apps might do this to trigger the permission request, as done by Apple's example code [22]. One app is the dictionary app Linguee, which runs a local server and communicates over the local WiFi IP address, triggering the permission. This could hint at a security issue, as Tang et al. [98] and Wu et al. [107] showed that apps can be vulnerable if they run wrongly configured servers accessible from other devices. For neither of the other 12 apps, we could find local network access in the traffic dump. While we found reports of requests to remote endpoints with port 10161 triggering the permission [105], we could not reproduce it on a current iOS version. We do not include those 14 apps as accesses to the local network as we could not observe any local network traffic, and the threats we analyze in this paper all require access to other devices on the local network.

Access Phases. We further analyzed when apps access the local network. More Android apps (75.21%) than iOS (65.13%) already accessed the local network before any user interaction. The iOS permission could have influenced those differences since it makes it transparent when apps access the local network for the first time. We found 70 apps that access it on both platforms: 13 Android apps (18.57%) access it before any interaction, while the corresponding iOS apps only do so after an interaction. In contrast, five iOS apps (7.14%) access the local network upon opening, while their Android counterparts do so only after an interaction. 42 apps (60%) access it on both platforms without interaction, and 10 apps (14.29%) only after an interaction.

TABLE 2: Number of apps that accessed the local network. The *Local* column indicates apps that sent a message to another local address, while *Broadcast* and *Multicast* show apps that sent respective messages. The *All* column provides the apps that used any of these methods. The numbers in the *Total* rows are relative to the dataset size. The shows apps that accessed the local network without any interaction, while those in only did so after an interaction. The percentages refer to their total number of accesses.

		All	Broadcast	Local	Multicast
	Total	152 (1.40%)	44 (0.41%)	18 (0.17%)	117 (1.08%)
		99 (65.13%)	22 (50.00%)	7 (38.89%)	84 (71.79%)
		53 (34.87%)	22 (50.00%)	11 (61.11%)	33 (28.21%)
	Total	117 (1.08%)	53 (0.49%)	24 (0.22%)	73 (0.67%)
		88 (75.21%)	34 (64.15%)	16 (66.67%)	63 (86.30%)
		29 (24.79%)	19 (35.85%)	8 (33.33%)	10 (13.70%)

Case Studies. We manually classified the apps that access the local network. Overall, most apps are companion apps for IoT devices (112, 56.28%), followed by video apps (17, 8.54%), and apps for events (16, 8.04%). We provide further details in Section A.2. We found apps from other categories less often, e.g., two shopping apps, a dating app, and a crypto wallet on Android access the local network. As we could not think of any use cases for them, we manually analyzed them. We observed local network access only in the Android version of all four apps and they all have Alibaba libraries in common. The apps perform an ARP scan of the connected WiFi after the app launches the first time. While they scan the network when the user opens them the first time, we did not observe this behavior consistently afterward. The code responsible for scanning is heavily obfuscated with encryption and reflection. We could not find any local network information in the traffic. Online users suspected AliExpress uses it for advertisement [83].

The TanTan dating app's privacy policy is the only one that relates to accessing the local network. They mention collecting "*device network information*," and information about the "*device environment*" [99]. We translated the policy using Google Translate, as the original page is Chinese. The fact that we did not find their counterpart iOS apps accessing the local network could be due to the permission, which makes the first access transparent, as there were also reports of the iOS AliExpress app accessing the local network in the past [83].

We performed another case study on the Among Us game app. It uses Universal Plug and Play (UPnP) to find other players. The app searches for other players immedi-

ately after opening on both platforms. If it finds any UPnP device in the network, it retrieves more information about it. In our test network, that were the Philips Hue bridge and Google Chromecast, which leak sensitive information. For example, the Philips Hue bridge returns the model name, model number, and serial number. All this information could be used by apps for ads and tracking.

Rationales and Bonjour Services. In our dataset, we found 727 (6.69%) apps with permission rationales, and 586 (5.39%) that declared a Bonjour string. Of those, 69 (11.77%) apps did not declare a rationale. Of all the iOS apps we found triggering the local network permission, 98 (61.25% of triggered permissions) apps declared a permission rationale, and 62 (38.75%) apps did not.

We discovered 27 apps that sent Bonjour mDNS messages but did not declare a Bonjour string, contradicting the documentation. We manually analyzed them and found that 8 (29.63%) apps were published before the string was required or targeted an older iOS version. We assume iOS does not enforce it for them for backward compatibility. 17 (62.96%) apps used a library for Google Cast that did not use any built-in Bonjour method. Instead, they relied on a custom implementation. iOS checks the Bonjour string only for the system methods. Thus, those apps bypassed this check. One app (3.7%) tried to communicate with a local domain name (domain name ending with `.local`). iOS internally uses Bonjour to find the IP addresses of local domains, but since the app did not use the Bonjour method, it also did not require the string. Lastly, the Philips Hue [92] app (3.7%), an IoT companion app, used a system method that requires the string but did not declare it. The app received updates frequently and targeted a current iOS version. We were not able to reproduce this with a test app and reported our findings to Apple in April 2024, who categorized it as *expected behavior* in August 2024.

4.3.2. Local Addresses Outside the Subnet. We found a similar number of iOS and Android apps that trying to contact local network addresses outside the connected local subnet. In total, we found 23 (0.21%) iOS and 22 (0.2%) Android apps exhibiting such a behavior. Nine apps did it across the platforms, 14 apps only on iOS, and 13 apps only on Android.

We found different reasons for this behavior: There were IoT companion apps that tried to contact their corresponding devices at a specific hard-coded address [89]. For example, the Android smart camera apps 4K CAM and Uniden 720 Link tried to reach their devices at `192.168.1.1` if the user did not input any address. Even though both apps were from different developers, they shared the same code for accessing the address, which could be a sign of a rebranded device. In other apps, we consider the contacted addresses to be development artifacts or bugs. For example, we observed that the Android app Door Entry CLASSE300X, belonging to a smart door lock and surveillance system, performed an ARP scan of a `/23` network space, even if connected to a `/24` network. The scan of the wrong network space happened because of a development bug when calculating the network mask. While the Android app scanned the local network after launch, we did not observe the corresponding iOS app accessing the local network as it did not access the local network before a user logged in, which could be due to the permission.

Takeaways

To answer RQ2: *How prevalent is the local network permission?*, we showed:

- 117 iOS and 152 Android apps in our dataset access the local network.
- Over half of the apps access the local network on start. iOS apps do this less often than Android apps, which could be an effect of the permission.
- Apps try to contact local addresses outside the connected subnet, highlighting the relevance of our finding regarding the limited protection of local network addresses in Section 3.

5. Permission Rationales

After understanding the technical aspects of the permission and the local network access of apps, we investigate users' ability to make informed decisions about the local network permission. When an app requests it, users are presented with a rationale composed of a system-generated description and a message that developers can set to explain why the app needs this permission. A default message is shown if developers do not provide a custom one. These messages are mostly all the information users get before deciding whether to grant the permission or not [38]. Investigating the content of these messages helps us understand what information goes into a user's decision-making process. Thus, we perform a content analysis on rationales to answer *RQ3: which concepts constitute the permission rationales?*

5.1. Methodology

As a first step, we automatically extracted rationales from iOS apps in our dataset, as already described in Section 4.2.3. We filtered these strings to only include the first rationale in German and English, as these are the languages in which all involved researchers are proficient. Furthermore, we used the Google Translate library [51] to translate the default language used by the app to English, leaving us with up to three rationales.

Next, three researchers independently coded a sample of 100 rationales. We aimed to identify the concepts rationales contain that we deem crucial for users' understanding to make an informed decision. One researcher coded all 100, while the other two coded 50 rationales each, meaning that two researchers coded each message. After this initial coding round, all researchers compared their codes, merged codebooks, and discussed potential disagreements.

All involved researchers agreed that most rationales follow a simple structure, often using similar compositions.

TABLE 3: Our codebook to categorize the rationales. We assigned a code if we found one of the keywords in a rationale. The column *# Apps* shows the number of rationales with the code, related to the total 727 apps with rationales.

Code	Keywords	# Apps
device interaction	device, devices, camera, cameras, gopro, speaker, speakers, tv, tvs, product, pc, pcs, iphone, ipad, server, servers, hngrynsite, computer, macos, car, vehicle, smart home, fritzbox, receiver, razorlink, printer, printers	539 (74.14%)
local network	local network, networks you use, lan, local area network	524 (72.08%)
discovery	discover, detecting, detect, discovering, search, find, finding, searching, scan, scanning, identify, identifying	371 (51.03%)
your network	your network, your wifi, your local network, your wi-fi, your local gateway, your gateway	348 (47.87%)
casting	cast-enabled, cast-capable, stream, cast-compatible, cast, chromecast-enabled, chromecast, screen mirroring	294 (40.44%)
WiFi network	wifi, wi-fi, wireless lan	275 (37.83%)
development artifact	debug, debugging, testing, developer, proxyman	32 (4.40%)
gaming	player, multiplayer, multi-player, in-game, game, two-player, players	30 (4.13%)
location dependent network	in-store, around you, nearby	25 (3.44%)
improve user experience	experience, improve, improved, enhance, enhanced, optimize, optimized, optimizing, improving, enhancing, personalized, personalize, stable connection	24 (3.30%)
network knowledge	tcp-network, tcp, udp, dns, voip, bonjour	22 (3.03%)
Internet connection	internet	4 (0.55%)
network quality testing	test the quality	3 (0.41%)

Also, all identified concepts are explicitly contained in the message, meaning that we can use a keyword-based approach to code our dataset [70]. Hence, we developed a keyword list for each code in our codebook. If a rationale contained one keyword from the list, we automatically assigned the corresponding code. Thus, one rationale can have multiple codes assigned. For example, we labeled the rationale "*The app uses the local network to set up and control your connected devices.*" with the codes *local network*, and *device interaction*, as it contains the keywords *local network*, and *devices*. We refined our keyword lists by looking into rationales for which no matching keyword was found and checking randomly-selected coded rationales. We iterated the processes of coding and refining until we found no new terms to add to the keyword lists, and only rationales without meaningful content remained uncoded. Further, to check that we did not miss prevailing concepts, we computed a count of each word in the rationale messages, which yielded no new concepts. We provide the full codebook and final list of keywords in Table 3 along with the number of rationales each code got assigned to. We used this codebook as the basis for a content analysis, with the goal of deriving the concepts constituting the permission rationales.

5.2. Results

In our dataset, 727 (6.69%) apps contain a permission rationale. We report the concepts we identified during the content analysis and their occurrences in rationales based on our keyword-based approach.

Local Network. The most prominent concept we encountered was that of a *local network*. 81.02% of apps with a rationale referred to some form of local network. We found this across all different types of apps and use cases. Rationales primarily used the terminology *local network* or the acronym *LAN*, while several also referred directly to the wireless variant, e.g., *WiFi* or *wireless LAN*. A large portion of apps, 348 (47.87%), used terminology implying some form of ownership of the network in question by including the word *your*, i.e., *your network* or *your WiFi*. A smaller portion, 28 (3.85%), used the phrase *networks you use*, which might point users towards the technical reality, where all networks a device is subsequently connected to are affected by the permission. Finally, 25 rationales (3.44%) employed terminology referring to physical proximity. Expressions such as *around you* and *nearby* suggest that devices close to the user's phone are somehow affected by the permission, even though proximity does not necessarily imply they are part of the same network.

Device Interaction. The second major concept permission rationales contained was *device interaction* with 539 mentions (74.14%). This comprises any kind of communication with another device, e.g., TVs, IoT devices, or phones. The most common use cases for this interaction were *discovering* other devices and *casting* video or audio. Often, these concepts appeared in conjunction. Other use cases were generic interactions of *transferring data* or simply connecting to other devices. However, if discovering other devices was given as the main reason for requesting the permission, we also often found that no specific follow-up interaction was described. That means it was unclear what the app was doing with the gathered information and users are left to infer the specific use case from the app's purpose.

Niche Use Cases. We further identified rationales referring to other use cases without including specific mentions of other devices. None of these use cases was present in more than 5% of rationales. The most frequent one was *gaming* (4.13%), i.e., searching for other players. 3.30% of apps promised an *improved user experience* if the permission was granted. Mostly, this was done without explaining how exactly it was accomplished. Four apps in our dataset stated to require the permission to access the Internet, which is not correct since iOS does normally not restrict Internet access. Finally, three rationales gave *testing the network quality* as their reason for requiring the permission.

Network Knowledge. 22 apps (3.03%) required some *network knowledge* to process, as they included technical abbreviations such as TCP, or VoIP. By looking at the provided text, it became apparent that eleven of these messages,

along with 21 others, were artifacts from development. This finding suggests that either the developers failed to remove these messages from the published version of their software, or that the app includes a dedicated debugging mode.

Takeaways

To answer RQ3: *Which concepts constitute the permission prompts?*, we showed:

- The most common concept is *local network*.
- To make an informed decision, users often need to understand what *casting* is, as well as which implications *discovering other devices* (i.e., scanning the network) can have.
- Alarmingly, the terminology in many rationales can be potentially misleading by limiting the scope (*your network*), creating false associations (*devices nearby*), or outright fueling misconceptions (*Internet access*).

6. Users' Permission Comprehension

After establishing the information users are presented with, we need to answer *RQ4: What is the user's understanding of these concepts?* We need to understand users' perception of the relevant concepts in order to establish their capability to make an informed decision. We investigate iOS users' knowledge of the permission and draw out their perception of the underlying technology using an online survey. Furthermore, we measure how widespread selected misconceptions are.

6.1. Methodology

As a necessary prerequisite to investigate iOS users' understanding of the local network permission, we derive the underlying concepts that are important for a correct understanding. Furthermore, we compose a list of common misconceptions to quantify how widespread they are and investigate users' awareness of common threats resulting from apps having local network accesses, as we discussed in Section 2.2.

6.1.1. Concepts Important for Understanding. We determined the fundamental concepts behind the local network permission through an expert brainstorming session. Three researchers met and discussed the results obtained from the content analysis of rationale messages (see Section 5.2). Two had a strong background in mobile security, the other one was experienced with usable security research. From this session, we derived the following concepts:

- **Boundaries:** a network's topology, i.e., which devices belong to the same local network, which are outside of it.
- **Transitivity:** the local network of a mobile device can change while it moves from place to place. The permission is granted once and then applies to all local networks.
- **Proximity:** devices physically close to each other are not necessarily part of the same network.

6.1.2. Misconceptions. To identify common misconceptions, three researchers independently conducted an online search on Google for the term *"local network permission."* We limited our search to the most commonly used platforms for tech-related problems: Reddit, StackOverflow, and X (formerly Twitter). The researchers then combined their findings into one coherent list. We include every misconception encountered at least once.

The most frequent beliefs we observed were that apps require the local network permission for using *Bluetooth* [103] or to access the *Internet via WiFi* [80], [104]. Some thought that apps could obtain the *WiFi password* [84] if granted the permission. Finally, people expressed that denying the permission can prevent other devices on the network from seeing their phone [82].

6.1.3. Survey Design. We used Qualtrics [76] to create our survey. We opted for a questionnaire consisting of mostly closed-ended questions. Doing so allows us to cover participants' knowledge of all relevant concepts, and to quantify how widespread misconceptions are. All that while still keeping the survey short to avoid fatigue.

Structure. The final questionnaire can be accessed online.[1] We first asked participants whether they knew what the local network is, and if they confirmed, we asked them to explain it in their own words. Next, we presented all participants with a multiple-choice question about the local network. A second multiple-choice question asked which use cases out of a given list would require an app to request the local network permission.

After this initial assessment of participants' general knowledge of local networks and the corresponding permission, we introduced a brief scenario, which serves as a reference for the remaining questions. For this, we designed a mock-up screenshot of a fictional app displaying the local network permission prompt, while showing the app's login screen in the background. A short text informed participants about the scenario, asking them to imagine that they just installed a new video-based social media app, which prompted them with the given permission request upon first launch.

We chose this setup as streaming to cast-enabled devices was a prominent use case in the permission rationales apps provided (see Section 5). We designed the scenario to be as realistic as possible. Hence, our mock-up app mimics a social media platform used by millions of people daily. We went with a fictional app to avoid biases towards existing systems, e.g., whether users trust or like an app. We included the default permission rationale message along with the system-generated description to establish a baseline.

After outlining the scenario as a reference for the remainder of the survey, we presented participants with a series of statements, divided into three blocks. For each statement, participants had to decide whether it is true or

1. Final survey questionnaire designed with Qualtrics [76] in our artifact: https://github.com/SecPriv/local_network/blob/main/survey.pdf

false. We also included an "I don't know" option, to discourage guessing. Each block began with a short reminder text about the scenario. The survey tool randomized the order of statements within each block for every participant to mitigate learning or unwanted side-effects.

The first block had participants imagine themselves in their own homes, connected to their WiFi. It consisted of nine statements: six covering a different threat model each, two covering misconceptions, and one filler statement. We made sure to include a mix of true and false statements to not give away the correct answer to participants. The second block continued the scenario with the user leaving their home and walking down the street, not being connected to any WiFi network. The three statements in this block covered the concepts of boundaries, transitivity, and proximity, respectively. In the third block we asked participants to imagine themselves sitting in a café or working in an office, with the phone being connected to the WiFi networks of these respective places. Of the six statements in this block, three concerned the concepts of transitivity, and two the concept of boundaries, while one was a filler statement.

Finally, we measured our participants' tech-savyness using the Affinity for Technology Interaction (ATI) scale [45] and collected demographic information, including technology background, and the iOS version they use. To ensure the quality of our dataset, we concluded the survey by asking participants if they had answered all questions carefully, and if they wanted their data to be included in our study. We accompanied this with a brief description of the importance of reliable data for scientific progress and assured participants that they will receive their compensation regardless of their response. In addition, we used the multiple-choice questions and the ATI scale as sanity checks, flagging participants who selected "non of the above" plus other items for the former, or chose 1 or 5 for all items on the latter.

6.1.4. Recruitment. We chose Prolific [74] as a service provider for recruiting participants, as it has shown to deliver high quality results in related work [96]. Before launching our study, we refined the survey through pilot testing. First, we asked friends and colleagues to review our questionnaire and worked out any misunderstandings. In this stage, we approached both tech-savvy individuals, as well as those without a technical background. Then, we advertised the study on prolific for a small sample of ten participants to pilot test in a realistic setting, offering 4£ for compensation. We did not include these responses in the final data set since we adapted one question based on the responses we received. We also used this second pilot test to get a solid estimate of the time it takes to complete. Finally, we opened our study on Prolific for 150 participants. The number is above 121 participants required for a 99.9% confidence interval assuming 50% of the population knows the correct answers, considering a 15% margin of error, which is in line with related work [65] on iOS permissions. We applied the following criteria to participants: (1) above 18 years of age, (2) uses iOS devices, and (3) currently resides in the US. We advertised the survey as a study on iOS permissions, without mentioning security or privacy to minimize selection bias. During our study, two people aborted the study before completing it. Prolific filled up the quota with additional participants. We offered 2£ of compensation for an estimated duration of 12 minutes.

Ethics. Our institution's Ethical Research Board (ERB) approved the study design. We only collect necessary data, and we store and process them in line with the General Data Protection Regulation (GDPR). Before starting the survey, we presented participants with a consent form detailing all information regarding data collection, their rights w.r.t. consent withdrawal, and the contact information of a responsible researcher. Only after explicitly providing consent, participants are forwarded to the survey.

6.1.5. Data Analysis. We evaluated multiple-choice questions by counting the number of correct answers. For single-choice questions, we counted how many participants answered them correctly, incorrectly, or responded with "I don't know." Some concepts and threats are covered by multiple questions. In those cases, we counted answers as correct if all questions were answered correctly, as incorrect if at least one was wrong, and as not sure otherwise. We evaluated all discrepancies between answers manually.

To draw comparisons, we divided participants into two groups, one with and one without knowledge of what a local network is. For grouping we used the answers to the question "Do you know what a local network is?" We asked participants who responded with "Yes" to briefly describe the concept in their own words. Based on these descriptions we redistributed participants to the group without knowledge. Two researchers independently grouped each open-ended response w.r.t. the participants' understanding: one of people with no or an incorrect understanding of local networks, and the other one with a correct understanding. We used Cohen's Kappa [55] to measure the inter-rater agreement. Afterward, we discussed and resolved disagreements.

To measure the influence of a person's understanding of what a local network is on their ability to correctly answer questions about it, we performed a T-test and calculated the effect size using Cohen's d. We performed χ^2 tests to study the differences between the groups and their answers regarding concepts, threats, and misconceptions.

For all tests, we stated the null hypothesis H_0 as "There is no difference between the two groups" and the alternative hypothesis H_1 as "There is a difference." We rejected H_0 for resulting p-values below the significance level of 0.05.

6.2. Results

First, we provide a general overview of our participants, followed by insights into their understanding of the local network and its permissions. We then detail common threats and misunderstandings. We provide an overview of our results and how they split up into participants with and without local network knowledge in Table 4.

TABLE 4: Results of our user study. The *total* columns reflect results from all 148 participants. The *local network knowledge* and *no knowledge* columns break down the data into groups based on whether we categorized participants as having an understanding of local networks. We related the numbers to the group sizes. For chi-squared tests marked with *, we focused only on correct and incorrect answers since fewer than five participants were unsure.

		Total ($N = 148$)			**Local Network Knowledge** ($N = 71$)			**No Knownledge** ($N = 77$)			χ^2 **Test**		
		Correct	**Wrong**	**Unsure**	**Correct**	**Wrong**	**Unsure**	**Correct**	**Wrong**	**Unsure**	χ^2	p	ϕ
Concepts	Boundaries	91 (61.49%)	24 (16.22%)	33 (22.30%)	42 (59.15%)	14 (19.72%)	15 (21.13%)	49 (63.64%)	10 (12.99%)	18 (23.38%)	1.24	0.54	0.09
	Proximity	88 (59.46%)	27 (18.24%)	33 (22.30%)	48 (67.61%)	12 (16.90%)	11 (15.49%)	40 (51.95%)	15 (19.48%)	22 (28.57%)	4.49	0.11	0.17
	Transitivity-1 Base	112 (75.68%)	24 (16.22%)	12 (8.11%)	65 (91.55%)	5 (7.04%)	1 (1.41%)	47 (61.04%)	19 (24.68%)	11 (14.29%)	*9.51	<0.01	0.26
	Transitivity-1	34 (30.09%)	64 (56.64%)	15 (13.27%)	20 (30.77%)	33 (50.77%)	12 (18.46%)	14 (29.17%)	31 (64.58%)	3 (6.25%)	*0.22	0.64	0.05
	Transitivity-2 Base	108 (72.97%)	28 (18.92%)	12 (8.11%)	60 (84.51%)	10 (14.08%)	1 (1.41%)	48 (62.34%)	18 (23.38%)	11 (14.29%)	*19.18	<0.01	0.14
	Transitivity-2	26 (23.85%)	55 (50.46%)	28 (25.69%)	15 (25.00%)	29 (48.33%)	16 (26.67%)	11 (22.45%)	26 (53.06%)	12 (24.49%)	0.24	0.89	0.05
Threats	Cross-user Tracking	75 (50.68%)	39 (26.35%)	34 (22.97%)	35 (49.30%)	19 (26.76%)	17 (23.94%)	40 (51.95%)	20 (25.97%)	17 (22.08%)	0.12	0.94	0.03
	Device Profiling	76 (51.35%)	35 (23.65%)	37 (25.00%)	45 (63.38%)	10 (14.08%)	16 (22.54%)	31 (40.26%)	25 (32.47%)	21 (27.27%)	9.46	0.01	0.25
	Exposing Devices	84 (56.76%)	20 (13.51%)	44 (29.73%)	41 (57.75%)	9 (12.68%)	21 (29.58%)	43 (55.84%)	11 (14.29%)	23 (29.87%)	0.10	0.95	0.03
	Location Profiling	74 (50.00%)	37 (25.00%)	37 (25.00%)	40 (56.34%)	18 (25.35%)	13 (18.31%)	34 (44.16%)	19 (24.68%)	24 (31.17%)	3.55	0.17	0.15
	At Least One	123 (83.11%)	21 (14.19%)	4 (2.70%)	62 (87.32%)	7 (9.86%)	2 (2.82%)	61 (79.22%)	14 (18.18%)	2 (2.60%)	*1.47	0.23	0.10
Misconcept.	Bluetooth	63 (42.57%)	73 (49.32%)	12 (8.11%)	28 (39.44%)	42 (59.15%)	1 (1.41%)	35 (45.45%)	31 (40.26%)	11 (14.29%)	*1.83	0.18	0.12
	Internet Access	48 (32.43%)	88 (59.46%)	12 (8.11%)	29 (40.85%)	41 (57.75%)	1 (1.41%)	19 (24.68%)	47 (61.04%)	11 (14.29%)	*1.86	0.17	0.12
	Visible Phone	32 (21.62%)	85 (57.43%)	31 (20.95%)	18 (25.35%)	42 (59.15%)	11 (15.49%)	14 (18.18%)	43 (55.84%)	20 (25.97%)	2.89	0.24	0.14
	WiFi Password	76 (51.35%)	24 (16.22%)	48 (32.43%)	39 (54.93%)	10 (14.08%)	22 (30.99%)	37 (48.05%)	14 (18.18%)	26 (33.77%)	0.81	0.67	0.07
	No Misconception	21 (14.19%)	125 (84.46%)	2 (1.35%)	9 (12.68%)	61 (85.92%)	1 (1.41%)	12 (15.58%)	64 (83.12%)	1 (1.30%)	*0.07	0.79	0.02

TABLE 5: Demographics of our final 148 participants, all currently reside in the US. This is after we excluded two participants from our dataset who did not want their data to be included or failed attention checks. We recruited a balanced sample in terms of gender, age, and IT background, with the majority running the latest iOS versions.

Demographics	**Participants (%)**
Female	89 (60.14%)
Male	57 (38.51%)
Non-binary	2 (1.35%)
18 - 24 years	35 (23.65%)
25 - 34 years	52 (35.14%)
35 - 44 years	35 (23.65%)
45 - 54 years	15 (10.14%)
55 - 64 years	8 (5.41%)
>= 65 years	3 (2.03%)
IT Background	72 (48.65%)
No Background	76 (51.35%)
iOS 17	96 (64.86%)
iOS 16	23 (15.54%)
iOS 15	9 (6.08%)
iOS 14	3 (2.03%)
iOS 13	5 (3.38%)
<= iOS 12	3 (2.03%)
Not Sure	9 (6.08%)
Permission Seen	116 (78.38%)
Permission Not Seen	16 (10.81%)
Not Sure	16 (10.81%)

6.2.1. Overview of Participants. We removed two participants from our study, resulting in a total number of 148 participants. One did not wish for their answers to be included, and one chose the same option for each item on the ATI scale, which indicates careless responding. On average, it took the participants 9 minutes and 58 seconds to fill out our survey. Most participants are between 25 and 34 years old (35.14%). The majority are female (60.14%), and almost half declared to have an IT background (48.65%). The mean ATI score is 3.46 ($sd = 0.87$), which is around the general population's average [45]. The most frequently used iOS version among participants is iOS 17 (64.86%), i.e., the latest version at the time of our study released in 2023, and 78.38% declared that they have seen the local network permission before. In Table 5, we provide further details on the demographics of our participants, their background and the iOS version they are using.

6.2.2. Local Network Understanding. Out of 148 participants, 31 (20.94%) stated that they do not know what the local network is, while the remaining 117 provided explanations. The inter-rater agreement on whether an explanation demonstrated a correct understanding of the concept was 0.82, considered almost perfect [55]. After reaching a consensus, we classified 71 responses as correct, leaving 46 incorrect responses, resulting in 77 participants without a correct understanding.

We tested whether our classification impacted the answers about local network knowledge and whether an IT background influenced the classification. With a one-sided T-test, we found a significant difference between the groups we classified and their score on the local network control questions ($p < 0.001$, $effect\ size = 0.88$), which supports the validity of our classification. Of the people with an IT background, we classified 58.33% as having a correct understanding, while for people without a background, the same was true for 38.16%. The χ^2 test shows a significant difference between the groups ($\chi^2 = 5.24$, $p = 0.02$, $\phi = 0.18$).

6.2.3. Concepts. We observed that the concept of *transitivity* is the most challenging for users to understand. First, we asked if a use case is possible at home after granting the permission, e.g., to discover other devices connected to the network. Later, we asked similar questions again in the café

and office scenarios. We evaluated if people who correctly answered the question in the home scenario recognized that access would also transfer to other local networks. We split up the evaluation into two parts, as we had two use cases for which we repeated the question in the scenarios. As the results are similar and the second is better suited for χ^2 tests, we only refer to those results, but both are included in Table 4 (in the Appendix). 72.97% of the participants answered the base question correctly. Only 23.85% had the follow-up question in other locations correct, and 50.46% answered at least one of them incorrectly. Having a local network understanding did not improve the ability to answer the transitivity concept correctly. The knowledge helped to get the base question right, 84.51% vs. 63.64% ($\chi^2 = 19.18$, $p < 0.01$, $\phi = 0.14$). However, when comparing the follow-up questions among those who answered the baseline correctly, both groups answered similarly. Of the group with knowledge, 25% answered it correctly, 48.33% wrong, and 26.67% were unsure, compared to 22.45% correct, 53.06% wrong, and 24.49% unsure in the other group.

More participants understood the concept of network *boundaries* and the related *proximity* concept. Overall, 61.49% answered the question about network boundaries correctly, while 16.22% answered it wrong. We asked two questions to test the understanding that devices in physically proximity do not have to be part of the same network. Both scenarios took place outside on the street, and we asked if the app could communicate with a smart TV inside a shop or another person's phone passing by. 59.46% correctly answered both questions, 22.30% were unsure what to answer, and 18.24% had at least one of the answers wrong. To improve the understanding, iOS could ask permission again for every new local network the app tries to access. This would make it transparent to the users to which local network the app has access to, allowing them to make individual decisions.

6.2.4. Threats. We asked participants about four threats coming from local network access: (1) *exposing devices*, (2) *inferring the location*, (3) *cross-user tracking*, and (4) *device identification*. On a positive note, for each threat, at least half of the participants displayed some sort of awareness. Most participants knew that devices protected by a firewall are still reachable from within the network, 56.76% correctly answered the respective questions, while 29.73% were unsure.

Further, we tested the knowledge about device detection within the local network with two questions. One question was general, while the other explicitly asked about sensitive devices, e.g., security cameras or health-related devices. In total, 51.35% could correctly respond. Additionally, 14.19% knew about the general case but were unsure about sensitive devices, and 9.46% were not sure about both questions. Finally, 50.68% correctly identified the possibility to perform cross-user tracking by linking users together based on local network information, 26.35% did not deem that possible while the rest were unsure.

We also asked two questions about inferring a users location: one about the possibility of obtaining the approximate location, e.g., through the router's MAC address, and one about inferring location changes based on differences in the connected network. Half of the participants got both questions right, and 25% had at least one answer wrong. Considering the latter, users seem to be more aware of the potential to estimate a location based on network information, as 13.51% correctly responded, but they were unsure about the other. In contrast, only 3.38% of respondents were unsure about the approximate location but knew of the possibility of tracking changes.

Device detection is the only threat for which we observed that users with local network knowledge did significantly better ($\chi^2 = 9.46$, $p < 0.01$, $\phi = 0.25$). 63.38% of the group with an understanding of the local network were aware of it, compared to 40.26% of those without. Positively, we observed that even without being able to explain the local network correctly, participants were aware of threats coming from apps having access. Even if the consequences might not be that obvious compared to other permissions, 83.11% of participants knew of at least one threat.

6.2.5. Misconceptions. Despite many of our participants having a technical background and an understanding of the local network, we observed that misconceptions are common. Nearly everyone (84.46%) had at least one misconception about the permission. The most common misconception is that apps require local network access to access the Internet (59.46%), closely followed by the belief that phones are not visible within the local network if the permission is denied (57.43%). Nearly half of the participants (49.32%) think that apps require it for Bluetooth communication. The only misconception less widespread is that the permission allows retrieving the WiFi password (16.22%).

Surprisingly, we observed that the group with local network knowledge did not perform significantly better. The misconception about Bluetooth was even more widespread (59.15% vs. 40.26%) among those with local network knowledge, while all others were similarly common: Internet access 57.75% vs. 61.04%, phone visibility 59.15% vs. 55.84%, and WiFi password 14.08% vs. 18.18%.

Rationales. Malicious apps could exploit misconceptions to trick users into accepting the permission by mentioning misconceptions in the rationales. However, exploring the effect requires further user studies. In Section 5, we found four apps mentioning *Internet* in their rationales, e.g., one app states "*This app requires an Internet connection to access your account data and vehicle features. Your local network will be used when cellular data is not available.*", which could trick users into giving permission. However, it could also be that the app accidentally triggers the permission, e.g., a library, and developers themselves suffer from the misconception. To avoid misconceptions, Apple could check if rationales contain keywords hinting towards misconceptions before releasing the app on the App Store.

Takeaways

To answer RQ4 *What is the user's understanding of these concepts?*, we showed:

- Even participants with a general understanding of the local network did not know about all the concepts required to make an informed decision.
- Misconceptions about the local network permissions are widespread. Developers could exploit this to trick users into accepting the permission.

We recommend prompting for permission on each distinct WiFi. This behaviour is more in line with users' expectations and would allow them to make better informed decisions.

7. Limitation and Future Work

Our work faces limitations and offers opportunities for future work. For the internal workings of the protected resources for network access, we rely on Apple's developer documentation. This might not cover all methods to access the local network and thus miss other bypasses. Future work could reverse engineer the iOS implementation of TCC to study the internal permission handling. It is yet unclear how Apple will handle the local network permission for browsers that are not based on WebKit, after being required to allow other engines in Europe due to the EU's Digital Markets Act [43], [73]. As we have found permission bypasses in webview-related components, this and other permissions' enforcement is worth exploring. Furthermore, Apple announced the extension of the local network permission in macOS 15 (released in 2024) [26], prompting questions about its enforcement on this platform as well.

Our large-scale local network access study has limitations inherent to dynamic analysis. Apps might detect that they are analyzed and behave differently than they normally would [87], [109]. Our dynamic interactor might not trigger functionalities that lead to local network access, e.g., functionality available only after login or verification. Thus, our numbers are a lower bound. We accept all permissions. However, declining permissions might lead to different behavior regarding local network accesses [77]. It also remains unclear if and what data apps share with external parties about the local network and the devices within it. During our manual analysis, we found apps that we suspect of doing so, but we could not confirm it due to obfuscation. Future work could leverage data flow analysis to track data coming from the local network.

Bonjour services, like AirPlay, do not trigger the permission, potentially confusing users as to why some functionality does not require permission [81]. Also, AirPlay can use Bluetooth to discover nearby devices [23] potentially influencing the Bluetooth misconception. Follow-up work could study the misconceptions further and their influences on users' decisions on granting permission in depth.

Finally, we focused our user study on iOS users. Studying Android users' understanding of the local network could gain further insights into the permission's impact and overall effectiveness, as well as on how to design a similar permission for Android. Finally, our sample of users residing in the US might hinder the generalizability to other user groups with diverse backgrounds, also in light of regional differences in terms of privacy regulations and awareness. One complementary direction for future work is the automated mining of iOS and Android app store reviews, which has shown promise in providing a broad picture of users' perceptions of apps' functionality [44] and [90].

8. Related Work

Local Network Access. Kuchhal and Li [60] performed a large-scale empirical investigation of websites' local network accesses. Other work [1], [12], [53] investigated how websites could scan the local network and attack connected devices. We investigate mobile apps instead of websites. We expect different access behavior for mobile apps: (1) there are legitimate use cases for it (e.g., companion apps controlling their IoT devices, or mirroring the screen to a monitor), and (2) in iOS, a permission guards the access.

Sivaraman et al. [94] demonstrated how forged apps could attack devices on the local network. Könings et al. [59] showed privacy implications of zero configuration protocols if they reveal device names by local network communication. In addition to user information, they can reveal sensitive device information, like the OS version, which attackers can use to find known vulnerabilities, as shown by Geneiatakis et al. [46]. Tang et al. [98] identified vulnerable UPnP libraries in 13 apps by analyzing network services in iOS apps and the reuse of vulnerable libraries. In contrast, we focus on the preceding step, which is the local network access of apps and the permission guarding it on iOS.

Girish et al. [48] analyzed how IoT devices and mobile apps communicate via the local network. By executing Android apps they detected app libraries that scanned the local network. Further, they showed that household fingerprinting and cross-device user tracking are possible using multicast protocols, like mDNS or UPnP. On the contrary, we study the permission on iOS, its impact, and the users' understanding of local network accesses.

Permission Analysis. Reardon et al. [78] found Android apps bypassing permissions with side and covert channels. To find those, they executed the apps in an instrumented environment. Yeke et al. [108] and Tileria et al. [102] identified cross-platform privacy leaks between smartwatches and Android apps for which the Android permission model is not sufficiently designed. In comparison, we focused on the local network permission, which does not exist on Android and differs from other permissions.

Previous work on permission rationales investigated mostly Android apps. Liu et al. [62] showed that especially permissions which are difficult to understand were insufficiently covered by rationales. Elbitar et al. [42] discovered that purpose messages can assist users with making informed decisions but also highlighted that the clarity of a message's content is highly subjective.

Tan et al. [97] studied the effect of developer-provided rationales on iOS in 2014. At the time of their study, most messages (98.3%) informed the users about benefits when granting the permission. Shen et al. [91] investigated to what extent permission rationales can help users understand the scope of a permission. They covered all available permissions on Android and iOS, which at the time did not include the local network permission. Their results suggest that only a small fraction of users can correctly infer an app's capabilities after granting a permission. Most recently, Mohamed et al. [65] studied the app tracking transparency permission rationales. They identified so-called dark patterns that apps use to trick users into granting permissions. In contrast, we study users' local network understanding and misconceptions of its permission, a prerequisite to analyzing dark patterns, which we leave for future work.

9. Conclusion

We identified two iOS components that can bypass the permission and that the protection for complex networks and Virtual Private Network (VPN) is insufficient. With our dynamic analysis, we showed that the permission potentially influences when apps access the local network. We found more iOS apps accessing the local network than Android apps, which is likely caused by Bonjour, a multicast protocol by Apple. Further, we found apps using Bonjour methods without declaring a service string, thus bypassing a requirement of their usage. With our content analysis of permission rationales, we show that it is vital to have an understanding of what a local network is to make sense of most messages. Building on that, we studied users' understanding of the permission, threats coming from local network access, and misconceptions. Positively, we found that nearly every participant (83.11%) was aware of at least one threat. However, misconceptions were even more widespread (84.46% had at least one), which could help malicious apps to trick users into granting permission.

Acknowledgments

We thank Paul Hager for his preliminary work on local network scanning on Android. This material is based on research supported by the Vienna Science and Technology Fund (WWTF) and the City of Vienna [Grant ID: 10.47379/ICT19056 and 10.47379/ICT22060], the Austrian Science Fund (FWF) [Grant ID: 10.55776/F8515-N], the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association, and SBA Research (SBA-K1), a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG.

References

[1] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based Attacks to Discover and Control Local IoT Devices," in *Proc. of the Workshop on IoT Security and Privacy (IoT S&P)*, 2018. DOI: 10.1145/3229565.3229568.

[2] A. A. Al Alsadi, K. Sameshima, J. Bleier, K. Yoshioka, M. Lindorfer, M. Van Eeten, and C. H. Gañán, "No Spring Chicken: Quantifying the Lifespan of Exploits in IoT Malware Using Static and Dynamic Analysis," in *Proc. of the 17th ASIA Conference on Computer and Communications Security (ASIACCS)*, 2022. DOI: 10.1145/3488932.3517408.

[3] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," in *Proc. of the 13th International Conference on Mining Software Repositories (MSR)*, 2016. DOI: 10.1145/2901739.2903508.

[4] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Z. Snow, F. Monrose, and M. Antonakakis, "The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle," in *Proc. of the 30th USENIX Security Symposium*, 2021.

[5] Amazon. "Amazon Alexa." Archived at https://archive.is/2aUEx. (2024), [Online]. Available: https://alexa.amazon.com/.

[6] Android Developers. "UI/Application Exerciser Monkey." Archived at https://archive.is/EuqsW. (Apr. 12, 2023), [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey.

[7] Android Developers. "Android Debug Bridge." Archived at https://archive.is/jJFBb, Version: 1.0.41. (Feb. 9, 2024), [Online]. Available: https://developer.android.com/tools/adb.

[8] Android Developers. "Manifest.permission." Archived at https://archive.is/LGldg. (Feb. 16, 2024), [Online]. Available: https://developer.android.com/reference/android/Manifest.permission#CHANGE_WIFI_MULTICAST_STATE.

[9] Android Developers. "Permissions on Android." Archived at https://archive.is/b21T7. (Mar. 7, 2024), [Online]. Available: https://developer.android.com/guide/topics/permissions/overview.

[10] Android Developers. "Request runtime permissions." Archived at https://archive.is/ujApe. (Apr. 1, 2024), [Online]. Available: https://developer.android.com/training/permissions/requesting.

[11] Android Developers. "WifiManager.MulticastLock." Archived at https://archive.is/SYYh7. (Feb. 16, 2024), [Online]. Available: https://developer.android.com/reference/android/net/wifi/WifiManager.MulticastLock.

[12] S. Antonatos, P. Akritidis, V. T. Lam, and K. G. Anagnostakis, "Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure," *ACM Transactions on Information and System Security*, vol. 12, no. 2, Dec. 2008. DOI: 10.1145/1455518.1477941.

[13] "Appium." Version: 2.4.1. (2024), [Online]. Available: http://appium.io.

[14] Apple. "About AirPlay." Archived at https://archive.is/msqDt. (Sep. 19, 2012), [Online]. Available: https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/AirPlayGuide/Introduction/Introduction.html.

[15] Apple. "SimplePing Example code." Archived at https://archive.is/7sEwR. (May 5, 2016), [Online]. Available: https://developer.apple.com/library/archive/samplecode/SimplePing.

[16] Apple. "If an app would like to connect to devices on your local network." Archived at https://archive.is/ld7Z8. (Oct. 24, 2020), [Online]. Available: https://support.apple.com/en-us/HT211870.

[17] Apple. "Should I use WKWebView or SFSafariViewController for web views in my app?" Archived at https://archive.is/YLX1c. (Oct. 8, 2020), [Online]. Available: https://developer.apple.com/news/?id=trjs0tcd.

[18] Apple. "Local Network Privacy FAQ-13." Archived at https://archive.is/CyGqd. (2021), [Online]. Available: https://developer.apple.com/forums/thread/663839.

[19] Apple. "Local Network Privacy FAQ-2." Archived at https://archive.is/Le8lh. (2021), [Online]. Available: https://developer.apple.com/forums/thread/663874.

[20] Apple. "Local Network Privacy FAQ-8." Archived at https://archive.is/2xaE5. (2021), [Online]. Available: https://developer.apple.com/forums/thread/663888.

[21] Apple. "Raw socket on iOS." Archived at https://archive.is/kYAj7. (2021), [Online]. Available: https://developer.apple.com/forums/thread/653072.

[22] Apple. "Triggering the Local Network Privacy Alert." Archived at https://archive.is/oPzYd. (2021), [Online]. Available: https://developer.apple.com/forums/thread/663768.

[23] Apple. "Use AirPlay with Apple devices." Archived at https://archive.is/5R8ov. (Oct. 27, 2021), [Online]. Available: https://support.apple.com/guide/deployment/use-airplay-dep9151c4ace.

[24] Apple. "Developer Documentation." Archived at https://archive.is/DKBNW. (2024), [Online]. Available: https://developer.apple.com/documentation.

[25] Apple. "Entitlements." Archived at https://archive.is/mep7b. (2024), [Online]. Available: https://developer.apple.com/documentation/bundleresources/entitlements.

[26] Apple. "Local Network Privacy FAQ." Archived at https://archive.is/rce8M. (Jul. 22, 2024), [Online]. Available: https://forums.developer.apple.com/forums/thread/663858.

[27] Apple. "NSLocalNetworkUsageDescription." Archived at https://archive.is/zlwj0. (2024), [Online]. Available: https://developer.apple.com/documentation/bundleresources/information_property_list/nslocalnetworkusagedescription.

[28] Apple. "Property List Key - NSBonjourServices." Archived at https://archive.is/FCSrW. (2024), [Online]. Available: https://developer.apple.com/documentation/bundleresources/information_property_list/nsbonjourservices.

[29] Apple. "Requesting access to protected resources." Archived at https://archive.is/Cin61. (2024), [Online]. Available: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/requesting_access_to_protected_resources/.

[30] Apple. "Requesting authorization to use location services." Archived at https://archive.is/L5MB9. (2024), [Online]. Available: https://developer.apple.com/documentation/corelocation/requesting_authorization_to_use_location_services.

[31] Apple. "Safari." Version: 8617.2.4.10.7. (2024), [Online]. Available: https://apps.apple.com/us/app/safari/id1146562112.

[32] Apple. "SFSafariViewController." Archived at https://archive.is/WVkDI. (2024), [Online]. Available: https://developer.apple.com/documentation/safariservices/sfsafariviewcontroller.

[33] Apple. "UIWebView." Archived at https://archive.is/tVkyn. (2024), [Online]. Available: https://developer.apple.com/documentation/uikit/uiwebview.

[34] Apple. "WKWebView." Archived at https://archive.is/KM4rb. (2024), [Online]. Available: https://developer.apple.com/documentation/webkit/wkwebview.

[35] S. Ashenbrenner. "Full Transparency: Controlling Apple's TCC." Archived at https://archive.is/v1s4s. (Jan. 16, 2024), [Online]. Available: https://www.huntress.com/blog/full-transparency-controlling-apples-tcc.

[36] P. Beer, M. Squarcina, L. Veronese, and M. Lindorfer, "Tabbed Out: Subverting the Android Custom Tab Security Model," in *Proc. of the 45th Symposium on Security & Privacy (S&P)*, 2024. DOI: 10.1109/SP54263.2024.00105.

[37] P. Beer, L. Veronese, M. Squarcina, and M. Lindorfer, "The Bridge between Web Applications and Mobile Platforms is Still Broken," in *3rd IEEE Workshop on Designing Security for the Web (SecWeb)*, 2022.

[38] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft, "Exploring Decision Making with Android's Runtime Permission Dialogs using In-context Surveys," in *Proc. of the 13th Symposium On Usable Privacy and Security (SOUPS)*, 2017.

[39] Brave Software. "Brave Browser: Private VPN." Version: 1.62. (2024), [Online]. Available: https://apps.apple.com/us/app/brave-browser-private-vpn/id1052879175.

[40] S. R. Choudhary, A. Gorla, and A. Orso, "Automated Test Input Generation for Android: Are We There Yet?" In *Proc. of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015. DOI: 10.1109/ASE.2015.89.

[41] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna, "Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis," in *Proc. of the 24rd Network and Distributed System Security Symposium (NDSS)*, 2017. DOI: 10.14722/ndss.2017.23465.

[42] Y. Elbitar, M. Schilling, T. T. Nguyen, M. Backes, and S. Bugiel, "Explanation Beats Context: The Effect of Timing & Rationales on Users' Runtime Permission Decisions," in *Proc. of the 30th USENIX Security Symposium*, 2021.

[43] European Parliament and the Council of the European Union, "Regulation (EU) 2022/1925 of the European Parliament and of the Council of 14 September 2022 on contestable and fair markets in the digital sector and amending Directives (EU) 2019/1937 and (EU) 2020/1828 (Digital Markets Act)," *Official Journal of the European Union*, vol. 65, L265 Oct. 2022. [Online]. Available: http://data.europa.eu/eli/reg/2022/1925/oj.

[44] M. Fassl, S. Anell, S. Houy, M. Lindorfer, and K. Krombholz, "Comparing User Perceptions of Anti-Stalkerware Apps with the Technical Reality," in *Proc. of the 18th Symposium On Usable Privacy and Security (SOUPS)*, 2022.

[45] T. Franke, C. Attig, and D. Wessel, "A Personal Resource for Technology Interaction: Development and Validation of the Affinity for Technology Interaction (ATI) Scale," *International Journal of Human–Computer Interaction*, vol. 35, 6 2019. DOI: 10.1080/10447318.2018.1456150.

[46] D. Geneiatakis, I. Kounelis, R. Neisse, I. Nai-Fovino, G. Steri, and G. Baldini, "Security and Privacy Issues for an IoT Based Smart Home," in *Proc. of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2017. DOI: 10.23919/MIPRO.2017.7973622.

[47] K. George. "Prices that change by the second: why shopping around for deals online isn't always worth it." Archived at https://archive.is/7mw9i. (Dec. 11, 2022), [Online]. Available: https://www.theguardian.com/lifeandstyle/2022/dec/12/prices-that-change-by-the-second-why-shopping-around-for-deals-online-isnt-always-worth-it/.

[48] A. Girish, T. Hu, V. Prakash, D. J. Dubois, S. Matic, D. Y. Huang, S. Egelman, J. Reardon, J. Tapiador, D. Choffnes, and N. Vallina-Rodriguez, "In the Room Where It Happens: Characterizing Local Communication and Threats in Smart Homes," in *Proc. of the 23rd Internet Measurement Conference (IMC)*, 2023. DOI: 10.1145/3618257.3624830.

[49] Google. "Chromecast with Google TV." (2024), [Online]. Available: https://store.google.com/us/product/chromecast_google_tv.

[50] Google. "Google Chrome." Version: 122.0.6261.62. (2024), [Online]. Available: https://apps.apple.com/us/app/google-chrome/id535886823.

[51] "Google Translate API for Python." Version: 4.0.0rc1. (2024), [Online]. Available: https://github.com/ssut/py-googletrans.

[52] HackTricks. "macOS TCC." Archived at https://archive.is/JITyX. (2024), [Online]. Available: https://book.hacktricks.xyz/macos-hardening/macos-security-and-privilege-escalation/macos-security-protections/macos-tcc/.

[53] M. Hazhirpasand and M. Ghafari, "One Leak is Enough to Expose Them All: From a WebRTC IP Leak to Web-based Network Scanning," in *Proc. of the 10th International Symposium on Engineering Secure Software and Systems (ESSoS)*, 2018. DOI: 10.1007/978-3-319-94496-8_5.

[54] J. Hendrix. "TikTok CEO Testifies to Congress." Archived at https://archive.is/NjL0L. (Mar. 24, 2023), [Online]. Available: https://www.techpolicy.press/transcript-tiktok-ceo-testifies-to-congress/.

[55] L. Hsu and R. Field, "Interrater Agreement Measures: Comments on Kappan, Cohen's Kappa, Scott's π, and Aickin's α," *Understanding Statistics*, vol. 2, Jul. 2003. DOI: 10.1207/S15328031US0203_03.

[56] IKEA. "IKEA Home smart app and TRÅDFRI gateway support." (2024), [Online]. Available: https://www.ikea.com/us/en/customer-service/product-support/app-gateway/.

[57] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding IoT Security from a Market-Scale Perspective," in *Proc.*

of the 29th Conference on Computer and Communications Security (CCS), 2022. DOI: 10.1145/3548606.3560640.

[58] Kaspersky. "Roaming Mantis implements new DNS changer in its malicious mobile app in 2022." Archived at https://archive.is/3x4cG. (Jan. 19, 2023), [Online]. Available: https://securelist.com/roaming-mantis-dns-changer-in-malicious-mobile-app/108464/.

[59] B. Könings, C. Bachmaier, F. Schaub, and M. Weber, "Device Names in the Wild: Investigating Privacy Risks of Zero Configuration Networking," in *Proc. of the 14th International Conference on Mobile Data Management*, 2013. DOI: 10.1109/MDM.2013.65.

[60] D. Kuchhal and F. Li, "Knock and Talk: Investigating Local Network Communications on Websites," in *Proc. of the 21th Internet Measurement Conference (IMC)*, 2021. DOI: 10.1145/3487552.3487857.

[61] R. Kumar, A. Virkud, R. S. Raman, A. Prakash, and R. Ensafi, "A Large-scale Investigation into Geodifferences in Mobile Apps," in *Proc. of the 31st USENIX Security Symposium*, 2022.

[62] X. Liu, Y. Leng, W. Yang, W. Wang, C. Zhai, and T. Xie, "A Large-Scale Empirical Study on Android Runtime-Permission Rationale Messages," in *Proc. of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018. DOI: 10.1109/VLHCC.2018.8506574.

[63] "Magisk: The Magic Mask for Android." Version: v26.4. (2024), [Online]. Available: https://github.com/topjohnwu/Magisk.

[64] Microsoft. "Microsoft Edge: KI-Browser." Version: 122.2365.56. (2024), [Online]. Available: https://apps.apple.com/us/app/microsoft-edge-ki-browser/id1288723196.

[65] R. Mohamed, A. Arunasalam, H. Farrukh, J. Tong, A. Bianchi, and Z. B. Celik, "ATTention Please! An Investigation of the App Tracking Transparency Permission," in *Proc. of the 33rd USENIX Security Symposium*, 2024.

[66] Mozilla. "Firefox: Private, Safe Browser." Version: 41973. (2024), [Online]. Available: https://apps.apple.com/us/app/firefox-private-safe-browser/id989804926.

[67] Mozilla. "Mozilla Location Service." Archived at https://archive.is/53NwD. (2024), [Online]. Available: https://location.services.mozilla.com/.

[68] M. Neugschwandtner, M. Lindorfer, and C. Platzer, "A View To A Kill: WebView Exploitation," in *Proc. of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2013.

[69] Opera. "Opera Browser." Version: 4.5.0. (2024), [Online]. Available: https://apps.apple.com/us/app/opera-browser-with-vpn-and-ai/id1411869974.

[70] A.-M. Ortloff, M. Fassl, A. Ponticello, F. Martius, A. Mertens, K. Krombholz, and M. Smith, "Different Researchers, Different Results? Analyzing the Influence of Researcher Experience and Data Type During Qualitative Analysis of an Interview and Survey Study on Security Advice," in *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, 2023. DOI: 10.1145/3544548.3580766.

[71] "palera1n." Version: v2.0.0-beta.8. (2024), [Online]. Available: https://palera.in/.

[72] Philips. "Hue Smart Lightning." (2024), [Online]. Available: https://www.philips-hue.com.

[73] A. Pradeep, Á. Feal, J. Gamba, A. Rao, M. Lindorfer, N. Vallina-Rodriguez, and D. Choffnes, "Not Your Average App: A Large-scale Privacy Analysis of Android Browsers," in *Proc. of the 23rd Privacy Enhancing Technologies Symposium (PETS)*, 2023. DOI: 10.56553/popets-2023-0003.

[74] "Prolific." (2024), [Online]. Available: https://www.prolific.com/.

[75] Python. "Generate and Parse Apple .plist Files." Version: 3.8. (2024), [Online]. Available: https://docs.python.org/3/library/plistlib.html.

[76] "Qualtrics." (2024), [Online]. Available: https://www.qualtrics.com/.

[77] J. Reardon. "The Curious Case of Coulus Coelib." Archived at https://archive.is/atF7n. (Apr. 6, 2022), [Online]. Available: https://blog.appcensus.io/2022/04/06/the-curious-case-of-coulus-coelib/.

[78] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System," in *Proc. of the 28th USENIX Security Symposium*, 2019.

[79] Reddit. "Why is Instagram asking for access to devices on a local network?" Archived at https://archive.is/3vE7w. (Dec. 13, 2020), [Online]. Available: https://www.reddit.com/r/techsupport/comments/kc67r3/why_is_instagram_asking_for_access_to_devices_on/.

[80] Reddit. "Access the Internet." Archived at https://archive.is/HPZXU. (May 4, 2021), [Online]. Available: https://www.reddit.com/r/MicrosoftTeams/comments/n4tkhs/comment/gwxgatv/.

[81] Reddit. "AirPlay." Archived at https://archive.is/Z9YsS. (Mar. 9, 2021), [Online]. Available: https://www.reddit.com/r/ios/comments/m0vgqu/did_google_just_bypass_local_network_permission.

[82] Reddit. "Other Devices See Phone." Archived at https://archive.is/mu4zq. (Mar. 9, 2021), [Online]. Available: https://www.reddit.com/r/ios/comments/m0vgqu/comment/gqb8v8c/.

[83] Reddit. "Why does AliExpress need access to my local network?" Archived at https://archive.is/Nqw48. (Mar. 10, 2021), [Online]. Available: https://www.reddit.com/r/Aliexpress/comments/m1u2en/why_does_aliexpress_need_access_to_my_local/.

[84] Reddit. "WiFi Password." Archived at https://archive.is/guiDq. (Apr. 11, 2022), [Online]. Available: https://www.reddit.com/r/Nest/comments/u0qgzw/comment/i49br82/.

[85] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions," in *Proc. of the 25th Network and Distributed System Security Symposium (NDSS)*, 2018. DOI: 10.14722/ndss.2018.23143.

[86] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic," in *Proc. of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016. DOI: 10.1145/2906388.2906392.

[87] A. Ruggia, D. Nisi, S. Dambra, A. Merlo, D. Balzarotti, and S. Aonzo, "Unmasking the Veiled: A Comprehensive Analysis of Android Evasive Malware," in *Proc. of the 19th ASIA Conference on Computer and Communications Security (ASIACCS)*, 2024. DOI: 10.1145/3634737.363765.

[88] E. Rye and D. Levin, "Surveilling the Masses with Wi-Fi-Based Positioning Systems," in *Proc. of the 45th Symposium on Security & Privacy (S&P)*, 2024. DOI: 10.1109/SP54263.2024.00239.

[89] D. Schmidt, C. Tagliaro, K. Borgolte, and M. Lindorfer, "IoTFlow: Inferring IoT Device Behavior at Scale through Static Mobile Companion App Analysis," in *Proc. of the 30th Conference on Computer and Communications Security (CCS)*, 2023. DOI: 10.1145/3576915.3623211.

[90] G. L. Scoccia, S. Ruberto, I. Malavolta, M. Autili, and P. Inverardi, "An Investigation into Android Run-time Permissions from the End Users' Perspective," in *Proc. of the 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2018. DOI: 10.1145/3197231.3197236.

[91] B. Shen, L. Wei, C. Xiang, Y. Wu, M. Shen, Y. Zhou, and X. Jin, "Can Systems Explain Permissions Better? Understanding Users' Misperceptions under Smartphone Runtime Permission Model," in *Proc. of the 30th USENIX Security Symposium*, 2021.

[92] Signify Netherlands B.V. "Philips Hue." Version: 4.39.0. (2024), [Online]. Available: https://apps.apple.com/us/app/philips-hue/id1055281310.

[93] C. Silva. "TikTok users' favorite moments from the TikTok congressional hearing." Archived at https://archive.is/gfkIV. (Mar. 23, 2023), [Online]. Available: https://mashable.com/article/tiktok-congressional-hearing.

[94] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smart-Phones Attacking Smart-Homes," in *Proc. of the 9th Conference on Security & Privacy in Wireless and Mobile Networks (WISEC)*, 2016. DOI: 10.1145/2939918.2939925.

[95] M. Steinböck, J. Bleier, M. Rainer, T. Urban, C. Utz, and M. Lindorfer, "Comparing Apples to Androids: Discovery, Retrieval, and Matching of iOS and Android Apps for Cross-Platform Analyses,"

in *Proc. of the 21st International Conference on Mining Software Repositories (MSR)*, 2024. DOI: 10.1145/3643991.3644896.

[96] M. Tahaei, R. Abu-Salma, and A. Rashid, "Stuck in the Permissions With You: Developer & End-User Perspectives on App Permissions & Their Privacy Ramifications," in *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, 2023. DOI: 10.1145/3544548.3581060.

[97] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner, "The Effect of Developer-Specified Explanations for Permission Requests on Smartphone User Behavior," in *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, 2014. DOI: 10.1145/2556288.2557400.

[98] Z. Tang, K. Tang, M. Xue, Y. Tian, S. Chen, M. Ikram, T. Wang, and H. Zhu, "iOS, Your OS, Everybody's OS: Vetting and Analyzing Network Services of iOS Applications," in *Proc. of the 29th USENIX Security Symposium*, 2020.

[99] TanTan. "Privacy Policy." Archived at https://archive.is/iNhVo. (Mar. 29, 2024), [Online]. Available: http://lp.tantanapp.com/and_play/?id=17.

[100] "TCPDUMP & LIBPCAP." Version: 4.9.3. (2024), [Online]. Available: https://www.tcpdump.org/.

[101] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862, Sep. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4862.

[102] M. Tileria, J. Blasco, and G. Suarez-Tangil, "WearFlow: Expanding Information Flow Analysis To Companion Apps in Wear OS," in *Proc. of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2020.

[103] Twitter (X). "Roxana Nasoi: Okay for Bluetooth connectivity..." Archived at https://archive.is/c5ZWb. (Sep. 26, 2021), [Online]. Available: https://twitter.com/roxananasoi/status/1442205247624159233.

[104] Twitter (X). "Seth Rice: YES = It uses your WiFi..." Archived at https://archive.is/svQvq. (Nov. 19, 2021), [Online]. Available: https://twitter.com/DrumLordJr/status/1461590761187655685.

[105] Unity Forum. "TcpClient.Connect() on iOS 14 triggers local network access permissions required." Archived at https://archive.is/Sd7u2. (Nov. 5, 2020), [Online]. Available: https://forum.unity.com/threads/tcpclient-connect-on-ios-14-triggers-local-network-access-permissions-required.1000701/.

[106] Y. Wang, X. Liu, W. Mao, and W. Wang, "DCDroid: Automated Detection of SSL/TLS Certificate Verification Vulnerabilities in Android Apps," in *Proc. of the Turing Celebration Conference (TUR-C)*, 2019. DOI: 10.1145/3321408.3326665.

[107] D. Wu, D. Gao, R. Chang, E. He, E. Cheng, and R. Deng, "Understanding Open Ports in Android Applications: Discovery, Diagnosis, and Security Assessment," in *Proc. of the 26th Network and Distributed System Security Symposium (NDSS)*, 2019. DOI: 10.14722/ndss.2019.23171.

[108] D. Yeke, M. Ibrahim, G. S. Tuncay, H. Farrukh, A. Imran, A. Bianchi, and Z. B. Celik, "Wear's my Data? Understanding the Cross-Device Runtime Permission Model in Wearables," in *Proc. of the 45th Symposium on Security & Privacy (S&P)*, 2024. DOI: 10.1109/SP54263.2024.00077.

[109] O. Zungur, A. Bianchi, G. Stringhini, and M. Egele, "AppJitsu: Investigating the Resiliency of Android Applications," in *Proc. of the 6th European Symposium on Security & Privacy (EuroS&P)*, 2021. DOI: 10.1109/EuroSP51992.2021.00038.

Appendix A.

A.1. iOS Permission Test (see Section 3)

We connected our phone to a `192.168.2.1/24` network, and a VPN (`10.1.0.1/24`) for the IPv4 tests. For IPv6 tests, we used a stateful `fd00::1/24` network. For each class from Table 1, we tested the addresses in Table 6.

TABLE 6: Tested IP addresses of our test app.

	IPv4	IPv6
Local	`192.168.2.1` (Router) `192.168.2.100` (No device)	`fd00::1fa2` (Laptop) `fd00::fa` (No device)

Local outside	`10.1.0.1` (VPN)
	`10.10.10.10`
	`192.168.1.1`
		`fd01::1`

Multicast	`224.0.0.69` (Unassigned) `224.0.0.251` (mDNS) `239.255.255.250` (SSDP)	`ff02::1` (All nodes) `ff02::2` (All routers) `ff02::18d` (Unassigned) ff02::fb (mDNS) ff05::c (SSDP)
Broadcast	`192.168.2.255` `255.255.255.255`	

A.2. App Categories (see Section 4)

We manually categorized the apps accessing the local network instead of using the store categories to be more precise, e.g., there exists no IoT category [57], [89]. In Table 7, we provide the categories and the number of apps.

TABLE 7: Categories of apps that accessed the local network. We summarized all categories with two or fewer apps in *Other*. and show the number of apps that only accessed it on one respective platform, and ∩ the apps that accessed the local network on both platforms. The numbers are in relation to the 199 apps that access it on at least one platform.

	Total			∩
IoT	112 (56.28%)	25 (12.56%)	28 (14.07%)	59 (29.65%)
Video	17 (8.54%)	11 (5.53%)	1 (0.50%)	5 (2.51%)
Events	16 (8.04%)	16 (8.04%)		
Audio	13 (6.53%)	7 (3.52%)	4 (2.01%)	2 (1.01%)
Games	12 (6.03%)	6 (3.02%)	5 (2.51%)	1 (0.50%)
Network	4 (2.01%)	1 (0.50%)	1 (0.50%)	2 (1.01%)
Fitness	3 (1.51%)	2 (1.01%)	1 (0.50%)	
Organization	3 (1.51%)	3 (1.51%)		
Shopping	3 (1.51%)	1 (0.50%)	2 (1.01%)	
Other	16 (8.04%)	10 (5.03%)	5 (2.51%)	1 (0.50%)

Appendix B. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper looks into the local network permission on iOS from both a technical perspective as well as the user's perspective. To this end, the paper investigates under which circumstances the permission is enforced, and makes a user study about what end-users think this permission does.

B.2. Scientific Contributions

- Addresses a Long-Known Issue
- Identifies an Impactful Vulnerability

B.3. Reasons for Acceptance

1) The authors found ways to bypass the permission dialogue but still access the local network from the app sandbox.
2) The authors made a large-scale study with 11k apps and how they use the local network permission.
3) The authors made a user study.

B.4. Noteworthy Concerns

1) The network analysis only considers the first 30 seconds of app usage and 25 automated interactions. It might miss complex interactions, such as login, verification, etc.
2) The paper mainly relies on publicly available documentation of the protected resource for network access, but, as acknowledged in Section 7, does not reverse engineer iOS internals of the permission handling behind that protected resource. Thus, there might be oversights in potential bypasses.
3) As acknowledged in Section 7, the demographics of the study are narrow.