Security Types for Usable Permissions Prompts

CHRISTINE GLASCOTT, University of Pittsburgh, USA
HEMANT GOUNI, Carnegie Mellon University, USA
JONATHAN ALDRICH, Carnegie Mellon University, USA

Permissions prompts are used extensively in modern computing systems to capture end-user privacy and security choices. However, prior work reveals a significant gap between usability and effective security. Applications must be subjected to manual review to ensure presented prompts are accurate. Moreover, repetitive prompts often overwhelm users, leading to confusion and habituation towards consent. In this paper, we demonstrate how to synthesize prior work in type systems for information flow with results from usable security to both strengthen and simplify permissions prompts. By representing permissions requests via types, the system can verify required functionality ahead of time, ensuring that displayed permission request UIs are accurate. Types can be further exploited to provide ahead-of-time information regarding permission requests, and when combined with results from the usable security literature, they unlock strategies for highlighting access choices, consolidating related prompts, and communicating severity in the prompt interface, thereby enhancing user understanding while reducing habituation.

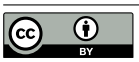
ACM Reference Format:
Christine Glascott, Hemant Gouni, and Jonathan Aldrich. 2026. Security Types for Usable Permissions Prompts. In *Proceedings of the 16th PLATEAU Workshop on Programming Languages and Human-Computer Interaction (PLATEAU '26)*. 10 pages. https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

1 Introduction

Useful programs often require access to sensitive user data [5–7]. This is often negotiated through interactive permission prompts served by the underlying application platform (like iOS, macOS, Android, or Windows) on the first instance of user data being requested by the application [14, 15]. The prompts provide context on why the information is needed along with potential risks [14, 15, 30]. Whether the program is allowed to access to the data is, in an ideal world, determined by the end user's pre-existing privacy and security preferences [14, 21, 30]. For example, if one downloaded a food delivery app, users would be prompted to decide whether they would like to transmit their location to the delivery driver for more expedient routing.

There are a variety of ways prompts can appear visually—in part because there is no one-size-fits-all option for all prompting scenarios [14]—but the majority tend towards offering *granular* choices. That is, each prompt presented to a user only focuses on a single request for a particular piece of data. While these fine-grained prompts can provide maximal control and contextual information for acting on a given data request [14, 28], the benefit is simultaneously limited by their verbosity. Studies found there to be lower comprehension rates for these fine-grained choices, from confusing terminology [15, 21] to repetition of similarly styled prompts [3, 19]. Repeated prompting plays an especially significant role in people's privacy decisions, rapidly becoming infeasible in assessing genuine security preferences by way of habituating users to answer affirmatively [4, 29, 30].

Authors' Contact Information: Christine Glascott, University of Pittsburgh, Pittsburgh, PA, USA; Hemant Gouni, Carnegie Mellon University, Pittsburgh, PA, USA; Jonathan Aldrich, Carnegie Mellon University, Pittsburgh, PA, USA.



PLATEAU '26, Pittsburgh, PA

https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

Habituation is the “decreased response to repeated stimulation” and is often determined to be a key factor in why users fail to remember prompt warnings [3, 4]. For instance, it has been shown that only 2 to 3 exposures to the same pop-up warning will prime the user to begin to “tune out” the information presented to them [3, 4]. The similarity in structure (as an allow/disallow prompt) and purpose (indicating security preferences) between these warning messages and permission prompts more broadly makes clear why the same issue arises throughout the latter [14, 19, 29].

On another note, software developers also struggle to create prompts that accurately convey what the platform truly needs to function. iOS requires developers to specify "a user-facing purpose string” to explain why the app needs that particular piece of data [7]. Similarly Android requires developers to declare each permission in the app’s manifest and if needed “explicitly list which of those permissions you are using and why” from API calls [5]. Yet, developers can make mistakes. Studies show developers often do not understand or even read guidelines for how to follow privacy and security regulations [8, 9, 20]. Further exacerbating the issue is that developers rarely prioritize user-centered explanations, operating under the assumption that users are uninterested in, or unable to understand, detailed privacy information [8, 9]. In addition, developers worry that users may interpret prompts as “something bad” [9] and abandon the platform, ultimately creating a tension between accommodating users’ privacy preferences and sufficiently expressive processes for communicating them.

Building off these concerns, a programming language designed with special support for permissions requests can offer prompts that precisely characterize what the system needs to function without overwhelming users, encouraging the latter to make more informed decisions. We propose a type system that decomposes a permissions request into pairs of sources and destinations, describing which data each request wishes to access—such as location information—and where that data should be sent—such as the company’s servers, an advertising service, or another program—followed by the options provided to the user in responding to the prompt. The type system then statically checks the source-destination pairs against the program, determining which information is being leaked and to where it’s being leaked using standard information flow techniques. It also accumulates a list of all possible permissions requests the platform may need to present to the user at run time, consisting of the cumulative source-destination information and metadata. This information is fed into heuristics derived from the usable security literature for simplifying permissions prompts. By joining similar requests, or altering initial requests to account for the knowledge of future requests, the total number can be curtailed. Any permissions requests ultimately seen by the user may have originated from a large number of coalesced and transformed prompts.

The general idea of grouping and bundling as strategies for minimizing permission prompts is not a new one. Multiple studies examining how users interact with privacy decisions suggest that structuring and grouping related permissions can help users stay focused and better understand the available choices [1, 14, 15]. Android developers use permission groups to “minimize the number of system dialogs that are presented to the user when an app requests closely related permissions” [5]. An interface design study with multiple privacy decisions at once found that grouping privacy settings helped users to navigate complex settings [1]. Caution must be exercised, however: for instance, Android permission groups have been known to result in overly broad permissions requests, encouraging developers to request more than what the application needs [10, 25].

To exploit the advantages of grouping permissions while avoiding its pitfalls, we build a variety of HCI design recommendations into the type system to inform how the permissions request data will be grouped and transformed to efficiently represent the available range of security choices while evading habituation. These same HCI heuristics will be used to influence the final presentation of the prompts on screen. Thus from start to finish, the type system verifies the core of each request, groups related requests, and outputs a finalized prompt tailored to the request scenario. This prompt

```
1 let user_location = secure_read(Location); // Pull metadata information
2
3 let google_socket = http.open("google.com"); // Destination defined
4
5 // => Prompt: "Would you like to share your location to Google?"
6 let request = req user_location google_socket [Once, WhileUsing, DontAllow];
```

Fig. 1. Logic for Initiating a Permissions Request

will be presented to users when the program executes. We describe in this work our in-progress designs and goals for exploiting the interplay between information flow typing and permissions prompting to make stronger guarantees of permissions prompts from the formal view and that of usability concerns.

2 Information Flow Typing

In order to maintain knowledge of precisely which data is being accessed by a permissions request, we deploy a standard theory of information flow typing [13, 23], in particular, *Structural Information Flow* [16]. This provides a simple regime for tracking the provenance of all data in the program, providing information at the point of a permissions request regarding precisely which data stands to be leaked. Permissions prompts are modeled as *declassification* within the information flow theory, that is, the intentional allowance of potentially insecure data flows.

Specifically, permissions requests appear in the program as expressions highlighting which data needs to be accessed and to which destination it will be sent along with any extra metadata—like the level of access being requested or a structured description of the reason for the request. For instance, the expression corresponding to a prompt like "Would you like to share your location with Google?" might look like that in Figure 1. We store a location on the first line, then open a socket to Google on the next. On line 6, where the keyword `req` is invoked to initiate a permissions prompt, the type system analyzes the first argument `user_location` to determine which information is being leaked to its second argument `google_socket`; in this case, it infers this is location data due its information flow dependency on `secure_read(Location)` on line 1. As its third argument, `req` is passed the options to appear in the permissions prompt.

The source-destination information produced by the type checker is used to inform the grouping of permissions requests and the final design of the presented prompt. This is done by placing each seen request into a "ready-to-prompt" list. Yet not all prompts in this list may need to be presented separately—indeed, this may ultimately be detrimental to obtaining accurate privacy choices from users—so we'd like to preprocess this list to collate requests. Multiple requests will be compared to find commonalities in function and purpose. Namely, those with similarities in source and destinations may be joined. The particular heuristics we deploy here will be discussed in section 3. Information is never lost throughout this process; joining permissions requests always results in a prompt which contains information from both. This will further inform different layout and visual styling strategies for the presented prompt to be discussed in section 4. Thus, the once multiple, individual requests are removed from the "ready-to-prompt" list and replaced by the now grouped request. Where an individual request within the group would have triggered, the grouped request now triggers instead and obviates the need for further prompting due to discharging a wider range of requests—which it knows would likely have been requested later—in one prompt.

3 Mechanizing Usable Security Guidance

Past research in usable security has observed a number of factors which play a key role in producing usable and secure permission prompts. There is widespread consensus that users need to understand *why* an application is asking for particular information [2, 15, 18, 21, 22, 24, 26]. The principle of *contextual integrity* suggests that it is essential for people to notionally understand the flow of information within an application to provide context regarding the appearance of a given prompt [22, 29, 30]. That is, modeling permissions prompts around the user's mental models of a program's data flows can significantly ease their privacy concerns and leads to higher comfort ratings [21].

Nevertheless, for the information contained within a prompt to be at all effective, user attention must remain focused on the prompt. This can be a challenge considering behavior surrounding permissions prompts has been found to be bifurcated between two primary kinds of users. The first are the *contextual users*, who fully interpret what is being asked of them and make careful decisions based on their particular preferences developed over the long term [1, 15, 22, 30]. The second are the *defaulter users*, who do not fully understand what is being asked of them and rely on simple heuristics and short-range preferences to quickly discharge permissions requests [1, 15, 22, 30]. The first kind of user prefers being given granular control over privacy preferences with a maximum of contextual information, whereas the second kind prefers fewer, and more succinct, prompts. We should aim to cater to both groups despite their opposite dispositions, making this a complex task.

To strike the correct balance between contextual clarity and simplified design, we will focus on automatically organizing and consolidating requests following recommendations from the literature. First, the source component of each request will be used to categorize into it into a permission group like app activity, app performance, calendar, camera, contacts, devices, files, financial, health and fitness, location, microphone, personal info, photos, messages/SMS, and videos [5, 20]. Request types falling within the same category will be grouped together (e.g., generic documents and audio files under files). These groupings follow the mental models of users and developers surrounding platforms like iOS and Android, with users preferring categorization by similar data types [1]. If there is a data type that has not already been categorized or does not fit under any of the defined groups above, it is up to the developer define a new group or allow that request to remain as an individual prompt. Observe the importance of sound information flow tracking to this simplification: we must be able to know that we are not producing inaccurately categorized permissions prompts, since this would lead to users being misled about what data they are releasing. Indeed, this assurance follows from typing. It will be essential, in general, that our simplifications operate on such soundly calculated information.

The next step is to assess requests based on their role in the function of the program. If two requests are in different permission category groups, but both are needed to achieve a certain function (broadly, being sent to the same or similar destinations), the requests can be joined together. In this process, neither of these requests will be moved under the other's permission category since we do not want to violate the meaning of the predefined category. Rather, the two requests will be formed into a new unique group separate from their original category. For instance, imagine a user downloads a mobile app that specializes in video conferencing. Our simplification procedure might recognize that there are two requests needed for interactive video chatting: microphone and camera access. It recognizes they originate from two different categories, but both are destined for the video call function. Thus, to align with the user's mental model of the reason for the request while minimizing habituation, these requests will be merged. This example will be elaborated further in section 4, where we discuss the resulting prompt UI.

We may not wish to apply the simplifications we have discussed thus far in all cases. For instance, it should be taken into account whether one request poses a higher risk than another.

We want to ensure that higher-risk requests (e.g., those for more sensitive data) are sufficiently distinguished from those of lower-risk. Moreover, users may be more protective of certain kinds of information than others. Studies have shown that users exhibit the highest privacy concern for data categories like financial information, and lesser concern for categories like app activity (e.g., browsing habits) [12, 27]. Based on established user privacy concerns, groups representing financial and personally identifying information are classified as high risk; health and fitness, location, contacts, microphone, camera, messages/SMS, calendar, photos, and videos are classified as moderate risk; and miscellaneous files, devices, app activity, and app performance are categorized as low risk. To take another example, imagine a user downloads a mobile fitness app for tracking physical activity like running, biking, etc. At runtime, users will be asked if they would like to share their location to track where they go and also to share their motion activity to track their speed and calculate health metrics for their body. Once more, the system recognizes these are two different permission categories that go to the same destination of the activity tracking function. It may seem plausible to merge them, but after checking the risk levels of each prompt, our simplification procedure avoids doing so. Health and fitness information, and location information, are both high risk data, meaning users should be allowed to make more granular choices for each [12, 27]. Therefore, in the case of the fitness app, there will be two separate prompts for each request.

Prompts' access choices stand to be improved as well. Often users are given options like "Allow Always," "Allow While In Use," and "Allow Once," or simpler binary choices like simply "Allow" and "Don't Allow". While "Allow Always" may seem like the simple choice to avoid being repetitively asked, it can lead to significant over-privileging, granting the application persistent access to sensitive data even when the user is not actively using a relevant feature, thereby violating the user's contextual integrity expectations [22, 29, 30] and allowing for the collection of data for secondary, often undesired, purposes [30]. Granting unrestricted access is reasonable only in limited circumstances, such as for a mapping application that will need location data even when the screen is not turned on. Presenting simpler binary choices yields even worse results in contributing to user habituation towards accepting prompts [15], due to being given a presumably all-or-nothing choice for a piece of the application's functionality. The milder "Allow Once" gives users the option to allow access to data only for a single instance, but this might not be the right choice if the function is repeatedly used and the prompt must be served repeatedly. As mentioned, only 2-3 repeated prompting appearances are needed before users exhibit habituation [3]; prompting every time is a well-known design mistake [30]. Our heuristics ensure that "Allow Once" will only be highlighted over other options if, given our knowledge of all the permissions requests the program may make, similar requests will not be re-made often. Roughly, if a prompt appears in the request list less than twice, highlight "Allow Once", if more than twice "Allow While in Use", if significantly more "Allow Always." We have also been exploring creating application-specific access options based on these statically observed usage patterns; for instance, that of camera access only being allowed for a particular video call is discussed in section 4.

Finally, we wish to handle appropriately merging structured descriptions of request justifications. In the naive case, this could be handled by concatenating the corresponding description strings with "and" in the middle. However, as hinted by calling them *structured* descriptions, they will be organized into a schema that allows them to be combined to produce more satisfying and concise results than the naive strategy suggested. We do not go into detail on this point, in part because we are still iterating on a sufficiently general design, but also because the description largely does not interact in an interesting way with the rest of the system (including typing and our simplification heuristics) beyond acting as extra metadata. Pitfalls of prior work include failing to help users make the correct security decisions because the description's scope is often misunderstood, with respondents believing it is either "narrower or broader" than its true meaning [15]. This can mean

that users do not understand whether access continues in the background or what specific data is being collected [2].

Drawing upon these findings, here is a summary of the usable security and privacy heuristics discussed that will influence groupings and permission prompt user interfaces:

(1) **Category:** If requests are a part of the same predefined permission category (e.g., 'files'), keep them grouped together to promote consistency and align with users' mental models of the platform. Granting a request originating from a specific category does not grant access to the whole, only to the specified data within it.
(2) **Function/Destination:** If requests, even from different categories, disclose data towards similar file paths or are broadly required to achieve a single user-facing function, group them together to align prompts with the user's notional understanding of the application.
(3) **Risk:** If one request presents a significantly higher risk than another, do not group the requests but allow the user to make separate, granular decisions for sensitive information. Moderate to low risk requests might be grouped together.
(4) **Access:** If the request is for access regularly needed for the application to perform properly (i.e., not a rare, one-off event), highlight "Allow While In Use" over "Allow Once" to reduce habituation while maintaining contextual control. Highlight "Allow Always" for requests that need background access.

4 Design and Implementation

After type checking and collating prompts, they are ready to be finalized into visual, interactive permission prompts. As discussed prior, comprehension is critical to how users interact with the prompt [2, 15, 22, 24], so simple wording and layouts are key [1, 11, 17]. Particularly for merged permission prompts, it is important that they faithfully convey their merged semantics to users. Thankfully, this is relatively easy—it turns out the simplification procedures have done the hard work already by ensuring prompt data remains faithful to the behavior of the program. All that remains is to follow some final guidance from the literature on how to present them.

We use the data points generated from the simplification procedure—namely, request category, relationship to functionality, and severity—to aid in generating the request UI. Permission categories, along with the destination the request data will be sent to, influence the title of the request and will help users identify which of their data is being leaked (and where to). This is the very first element of the UI seen by users and heavily influences their choices [1, 11, 17], so it's important to be direct and to-the-point here. The simplistic nature of the request categories from the prior section makes this easy. The request description and options are populated straight from the request metadata, and may be decorated by e.g. highlighting certain options in the request UI (owing to the heuristics for optimizing access level choices described prior).

Let's walk through the example shown in Figure 2. Consider the user experience when viewing the permission requests for a video conferencing app: the current approach on mobile platforms typically displays two separate prompts which may or may not be faithful to the program's actual functioning—one for camera access and one for microphone access (demonstrated by the two prompts on the left). These prompts appear back-to-back, have a very similar visual structure, and crucially, lack clarity regarding the scope of "Allow," leaving users confused about whether access continues in the background or is limited only to when they are actively on a call. This repetition and ambiguity create a frustrating and interruptive experience for users. In contrast, the proposed system (shown on the right) uses a single request that groups the camera and microphone requests due to their shared function (the "Call" feature). The description succinctly describes the purposes of both requests, and the access choices are specialized to the application's precise usage of the data

(while the app is running, only for the singular flow to the call being initiated, or no access). The second design offers specific choices which both enhance user control and comprehension while minimizing prompt fatigue. In addition to being more usable, it has also been validated against the program's behavior; so the added complexity of the prompt does not carry with it the potential for security bugs or increased burden on app platform reviewers.

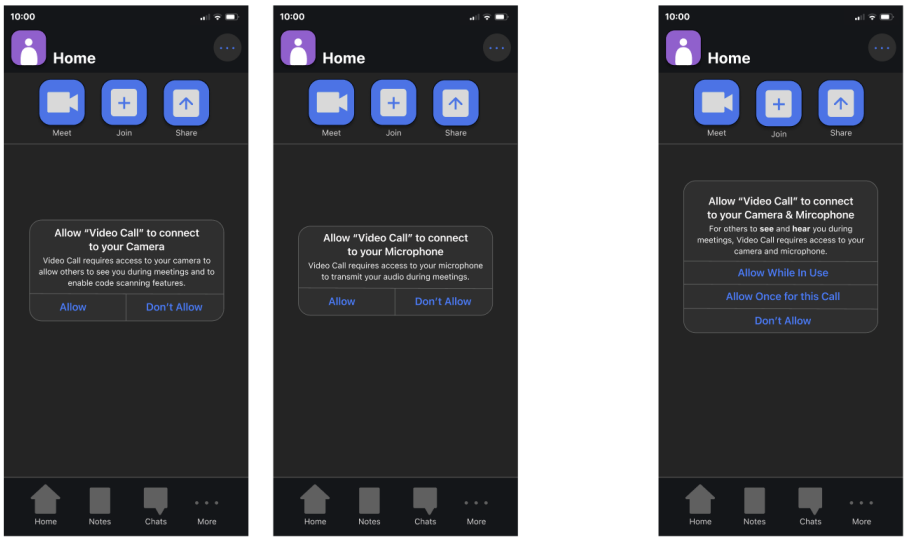


Fig. 2. This is the interface of an example app called Video Call for online video conferencing. The left two wire frames demonstrate how a non-typed requests would prompt users and the right wire frame demonstrates how a typed request would prompt users.

This example may seem quite straightforward, so let's consider another example where it's less clear how the merging strategies apply. Consider a mobile app called Drive for managing and storing files on your device, where one of its core features is connecting information stored on other devices a user may own. The three examples on the left side again represent the current approach using conventional requests: first, a prompt to enable Bluetooth; second, a prompt to sync files from the user's iPad; and third, a prompt to sync files from the user's personal computer, all appearing sequentially and causing interruption and fatigue. The user is also only given the options to "Allow" or "Don't Allow," which conveys little information about the nature of the application's access. Consistently in the background? Just for the one file transfer? It's not clear. To the right side is the single, typed, verified request that addresses these issues by grouping the Bluetooth activation, iPad sync, and PC sync requests together under the singular function of "Sync External Files." This consolidated prompt clearly articulates its purpose and offers comprehensive access choices—such as "Allow While In Use" and "Allow Only for This Sync" to enhance user control and communicate mental models of the kinds of permissions being granted while minimizing interruptions in using this multi-step file synchronization feature.

Ultimately, the goal of this implementation is to offer unique, intuitive designs for system requests. There is no one-size-fits-all solution for permission prompts, so it is critical to capture information that enhances not only user privacy awareness but also the overall user experience. The annoyance and frustration that current repetitive and vague prompts cause can lead to habituation, effectively blocking users from making meaningful privacy decisions. This implementation is therefore an

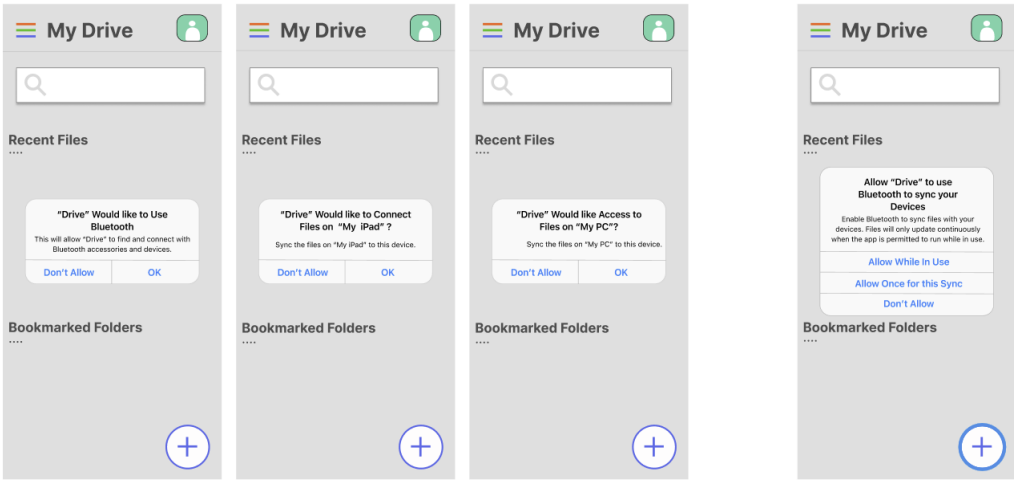


Fig. 3. This is the interface of an example app called Drive for file storage. The left three wire frames demonstrate how a non-typed requests would prompt users and the right wire frame demonstrates how a typed request would prompt users.

attempt to bridge the gap between usability and security, an endeavor from which developers also benefit. Our system attempts to maintain both the faithfulness of the prompts to the program while empowering the end user to make better choices, exploiting the interplay between these two concerns.

5 Limitations and Future Work

While we've presented our initial ideas for prompt visualization and illustrated the effectiveness of our heuristics in a few scenarios, the precise timing and context in which these grouped prompts appear is another critical factor mentioned in the literature that remains to be explored and implemented in this design. Most importantly, these designs must be empirically tested in a live environment. To facilitate this, future goals for this work include designing and implementing a live website for users to interact with as part of a user study. This includes fleshing out the design of our type checking procedure, heuristics, and generated prompts more fully, to enable eventual implementation in a realistic setting. Accordingly, we wish to explore many more scenarios to further refine the heuristics for checking and grouping processes, which will necessitate additional literature review and platform exploration beyond the limited scope discussed in this work.

References

[1] Eman Alashwali, Andrea Miller, Alexandra Nisenoff, Kyerra Norton, Ellie Young, Prerna Bothra, Mandy Lanyon, and Lorrie Faith Cranor. 2025. Interface Design to Support Informed Choices When Users Face Numerous Privacy Decisions. *IEEE Transactions on Privacy* 2 (2025), 79–91. doi:10.1109/TP.2025.3592580

[2] Nourah Alshomrani, Steven Furnell, and Ying He. 2023. Assessing User Understanding, Perception and Behaviour with Privacy and Permission Settings. In *HCI for Cybersecurity, Privacy and Trust*, Abbas Moallem (Ed.). Springer Nature Switzerland, Cham, 557–575.

[3] Ben B. Anderson, Jeffrey L. Jenkins, Anthony Vance, C. Brooke Kirwan, and David Eargle. 2016. Your memory is working against you: How eye tracking and memory explain habituation to security warnings. *Decision Support Systems* 92 (2016), 3–13. doi:10.1016/j.dss.2016.09.010

[4] Ben B. Anderson, Anthony Vance, C. Brooke Kirwan, David Eargle, and Jeffrey L. Jenkins. 2016. How users perceive and respond to security messages: a NeuroIS research agenda and empirical study. *European Journal of Information*

Systems 25 (2016), 364–390. Issue 4. doi:10.1057/ejis.2015.21

[5] Android Developers. 2025. Download Android Studio & App Tools. https://developer.android.com/studio. Accessed: 12-December-2025.

[6] Apple Developer. 2025. App Review. https://developer.apple.com/distribute/app-review/. Accessed: 12-December-2025.

[7] Apple Developer. 2025. Requesting Access to Protected Resources. https://developer.apple.com/documentation/uikit/requesting-access-to-protected-resources. Accessed: 12-December-2025.

[8] Rebecca Balebako and Lorrie Cranor. 2014. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security & Privacy* 12, 4 (2014), 55–58. doi:10.1109/MSP.2014.70

[9] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. doi:10.14722/usec.2014.23006

[10] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. 2010. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '10)*. Association for Computing Machinery, New York, NY, USA, 73–84. doi:10.1145/1866307.1866317

[11] Elijah Robert Bouma-Sims, Megan Li, Yanzi Lin, Adia Sakura-Lemessy, Alexandra Nisenoff, Ellie Young, Eleanor Birrell, Lorrie Faith Cranor, and Hana Habib. 2023. A US-UK Usability Evaluation of Consent Management Platform Cookie Consent Interface Design on Desktop and Mobile. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 163, 36 pages. doi:10.1145/3544548.3580725

[12] Hui Na Chua, Jie Sheng Ooi, and Anthony Herbland. 2021. The effects of different personal data categories on information privacy concern and disclosure. *Computers & Security* 110 (2021), 102453. doi:10.1016/j.cose.2021.102453

[13] Dorothy E. Denning. 1976. A lattice model of secure information flow. *Commun. ACM* 19, 5 (May 1976), 236–243. doi:10.1145/360051.360056

[14] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. 2012. How to Ask for Permission. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Security (HotSec '12)*. USENIX Association, Bellevue, WA, USA. https://www.usenix.org/conference/hotsec12/workshop-program/presentation/Felt

[15] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (Washington, D.C.) *(SOUPS '12)*. Association for Computing Machinery, New York, NY, USA, Article 3, 14 pages. doi:10.1145/2335356.2335360

[16] Hemant Gouni, Frank Pfenning, and Jonathan Aldrich. 2025. Structural Information Flow: A Fresh Look at Types for Non-interference. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 414 (Oct. 2025), 27 pages. doi:10.1145/3764116

[17] Hana Habib, Megan Li, Ellie Young, and Lorrie Cranor. 2022. "Okay, whatever": An Evaluation of Cookie Consent Interfaces. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 621, 27 pages. doi:10.1145/3491102.3501985

[18] Yousra Javed and Mohamed Shehab. 2017. Look before you Authorize: Using Eye-Tracking to Enforce User Attention towards Application Permissions. *Proceedings on Privacy Enhancing Technologies* 2017 (04 2017). doi:10.1515/popets-2017-0014

[19] Farzaneh Karegar, John Sören Pettersson, and Simone Fischer-Hübner. 2020. The Dilemma of User Engagement in Privacy Notices: Effects of Interaction Modes and Habituation on User Attention. *ACM Trans. Priv. Secur.* 23, 1, Article 5 (Feb. 2020), 38 pages. doi:10.1145/3372296

[20] Tianshi Li, Lorrie Faith Cranor, Yuvraj Agarwal, and Jason I. Hong. 2024. Matcha: An IDE Plugin for Creating Accurate Privacy Nutrition Labels. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 1, Article 33 (March 2024), 38 pages. doi:10.1145/3643544

[21] Jialiu Lin, Shahriyar Amini, Jason I. Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing *(UbiComp '12)*. Association for Computing Machinery, New York, NY, USA, 501–510. doi:10.1145/2370216.2370290

[22] Ricardo Mendes, André Brandão, João P. Vilela, and Alastair R. Beresford. 2022. Effect of User Expectation on Mobile App Privacy: A Field Study. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 207–214. doi:10.1109/PerCom53586.2022.9762379

[23] Andrew C. Myers. 1999. *Mostly-static decentralized information flow control.* Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA. doi:1721.1/16717

[24] Viktorija Paneva, Verena Winterhalter, Franziska Augustinowski, and Florian Alt. 2025. User Understanding of Privacy Permissions in Mobile Augmented Reality: Perceptions and Misconceptions. 9, 5, Article MHCI037 (Sept. 2025), 17 pages. doi:10.1145/3743738

[25] Sai Teja Peddinti, Igor Bilogrevic, Nina Taft, Martin Pelikan, Úlfar Erlingsson, Pauline Anthonysamy, and Giles Hogben. 2019. Reducing Permission Requests in Mobile Apps. In *Proceedings of the Internet Measurement Conference* (Amsterdam,

Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 259–266. doi:10.1145/3355369.3355584

[26] Vera Schmitt, Maija Poikela, and Sebastian Möller. 2022. Android Permission Manager, Visual Cues, and their Effect on Privacy Awareness and Privacy Literacy. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '22)*. Association for Computing Machinery, New York, NY, USA, Article 153, 12 pages. doi:10.1145/3538969.3543790

[27] Anya Skatova, Jaspreet Johal, Robert Houghton, Richard Mortier, Neelam Bhandari, Tom Lodge, Christian Wagner, James Goulding, Jon Crowcroft, and Anil Madhavapeddy. 2013. Perceived risks of personal data sharing.

[28] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. 2014. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 91–100. doi:10.1145/2556288.2557400

[29] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 499–514. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wijesekera

[30] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2017. The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences . In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1077–1093. doi:10.1109/SP.2017.51