Protecting Visual Information in Augmented Reality from Malicious Application Developers

Jk Jensen, Jinhan Hu
Arizona State University
Tempe, Arizona
jkjense3,jinhanhu@asu.edu

Amir Rahmati
Stony Brook University
Stony Brook, New York
amir@cs.stonybrook.edu

Robert LiKamWa
Arizona State University
Tempe, Arizona
likamwa@asu.edu

ABSTRACT

Visual applications – those that use camera frames as part of the application – allows for a rich, context-aware experience. The continuing development of mixed and augmented reality (MR/AR) on head-mounted displays (HMDs) furthers the richness of this experience by providing users a continuous vision experience, where visual information continuously provides context, and the real world is augmented by the virtual. However, these visual applications raise serious privacy concerns because they can capture private user information. To understand user privacy concerns in continuous vision computing environments, we study three MR/AR applications (augmented markers, augmented faces, and text capture). We show that in modern mobile visual applications, typical users are exposed to potential mass collection of sensitive information.

To address such deficiencies, we develop a framework that provides resource isolation between user information contained in camera frames and application access to the network. We implement the design as a proof of concept on the Android operating system and demonstrate its performance and usability with a modern state-of-the-art augmented reality library and several augmented reality applications. By comparing the applications from our case study with modified versions which better protect user privacy, results show that our design efficiently protects users against data collection in MR/AR applications with less than 0.7% performance overhead.

CCS CONCEPTS

• **Security and privacy** → *Mobile platform security*.

KEYWORDS

Operating system; Visual information protection; User privacy; Resource isolation; Split process

ACM Reference Format:
Jk Jensen, Jinhan Hu, Amir Rahmati, and Robert LiKamWa. 2019. Protecting Visual Information in Augmented Reality from Malicious Application Developers. In *The 5th ACM Workshop on Wearable Systems and Applications (WearSys'19), June 21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3325424.3329659

WearSys'19, June 21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6775-2/19/06…$15.00
https://doi.org/10.1145/3325424.3329659

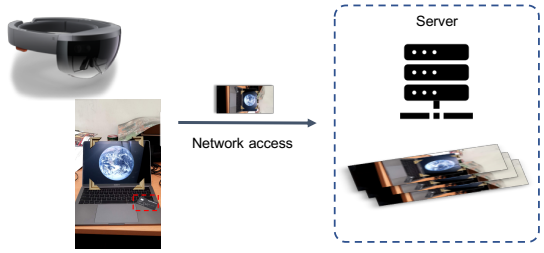


Figure 1: Camera frames in MR/AR often contain sensitive information (e.g. a credit card) which are vulnerable to collection by the application developer.

1 INTRODUCTION

As mobile computing evolves, its applications on HMDs shift to include a broader and deeper level of functionality, from image-guided minimally invasive procedures with Microsoft HoloLens [18] to instant social media post with Snap Spectables [7]. The growth of visual perception applications on mobile devices, which include those that utilize the on-device camera to enable rich user experiences, such as photography tools, social media sharing applications, and mixed and augmented reality experiences, allows for new lifestyles and opportunities.

However, visual applications raise serious privacy concerns and incur new challenges to protect user privacy because visual applications often contain private and sensitive user information that can be utilized maliciously by application developers, such as the credit card information accidentally captured in surroundings at fine level of detail. Even though mobile operating systems, e.g., Android OS or iOS, are equipped with permission systems, they still fail to protect user privacy in various scenarios, as developers can invalid them easily by mixing app code with non-trustworthy code such as advertisement libraries [2]. Wearable MR/AR applications are particularly vulnerable because application developers have unrestricted continuous access to sensitive visual information without transparency as to how the visual information is used [3, 5, 16].

We conduct a case study around three MR/AR applications running on the ARCore framework [11] which is one of the commercial platforms to provide environmental understanding, motion tracking, and light estimation for many MR/AR applications. The applications are *augmented markers*, which renders 3D objects above the markers (as shown in Figure 1); *augmented faces*, which adds effects to users' faces; and *text capture*, which digitalizes text seen through the camera. We find that in these scenarios **private user data are vulnerable to collection by the application developer**. While

providing a seemingly benign user experience, these applications may include hidden functionality to send visual information to a local server when a marker, a face, or a text is detected accordingly. The threat model we address in this work is demonstrated by these applications. We design for cases in which application developers can obtain sensitive visual information from the user and collect it off-device. We assume that the operating system is trusted and the developer does not utilize covert channels of communication, e.g., communication between two entities by manipulating shared resources [19].

Privacy concerns on modern computing systems, including inappropriate data collection in MR/AR applications, have been actively discussed in recent years. Related works address these privacy concerns by taint-tracking and controlling information flow [8–10, 14, 22] or adding an intermediate layer to preprocess frames and filter out sensitive visual information [4, 12, 20, 21]. However, they add inhibitive complexity and performance overhead which limits their practicality for real-world MR/AR applications.

We propose a MR/AR-specific development framework that provides resource isolation between user information contained in camera frames and application access to the network such that malicious applications developers cannot collect them off-device. We aim to raise barriers against visual information collection by guiding application development and enforcing information flow policies. Our framework is designed to protect against visual information leaks at the highest level of granularity with the most visibility – application shared resources. We also focus on developer freedom by not locking the developer into one vision framework or library. This is practical for actual MR/AR applications, which use a variety of computer vision platforms for visual computing, including in-house variations.

We evaluate our framework in the Android environment. However, the principles we proposed can also be applied to other platforms such as Windows because applications running on it are also in one process. By monitoring *network traffic* using the Android Studio profiler and *frame rate* reported by ARCore, results show that our framework effectively prevents malicious network access with negligible performance overhead (less than 0.7%).

We make the following contributions.

- We prove that MR/AR applications running on current mobile operating systems are vulnerable to data collection by malicious application developers.
- We introduce a framework to effectively isolate device camera frames from the network.
- We demonstrate that our framework can protect MR/AR application users against data collection in real-time with negligible overhead.

2 RELATED WORK

Information flow control Krohn et al. introduced the Flume system to allow safe interaction between conventional and DIFC-aware (Decentralized Information Flow Control) processes [14]. Enck et al. presented TaintDroid to identify and protect data leakage via untrusted applications [9]. Fernandes et al. introduced FlowFence to guarantee that sensitive data is only processed within designated functions that run in FlowFence-provided sandboxes [10]. Roy et al. presented Laminar which implements and further optimizes DIFC for program objects and OS resources [22]. Efstathopoulos et al. used Asbestos labels to isolate user data for information flow control [8]. However, each of these approaches trades performance for security which is impractical for performance-sensitive applications such as those in the MR/AR domain. Our framework utilizes a domain-specific approach that significantly restricts information flow from the camera to the network but with negligible overhead.

Protection of visual data Jana et al. presented the Darkly system which hides visual information from the developer by using opaque handles to operate on rather than the actual camera frame [12]. Roesner et al. introduced a framework that has a granularity to manage objects in the sensing streams [21]. Lebech et al. introduced Arya system to secure the rendering of AR applications [15]. Aditya et al. presented the I-Pic platform for policy-compliant image capture [4]. Lehman et al. developed PrivacyManager to help developers control malicious functionalities in AR applications [17]. Raval et al. proposed the MarkIt framework to allow users to specify and enforce fine-grained control over video feeds [20]. All of these systems require an intermediate layer to process visual information before the policies are applied, which allows for varying privacy granularity, but at the cost of complexity and overhead. Designing for a threat model which designates all visual information as sensitive removes the need for visual processing as part of the policy enforcement process. Removing the intermediate visual processing layer improves performance and removes complexity.

Policy Enforcement Jia et al. proposed an approach to enforce information flow control policies at runtime [13]. Tuncay et al. introduced Cusper to allow applications to change permissions dynamically [23]. ComDroid analyzes intent statically [6]. Laminar enforces the security policies at runtime [22]. The design of our framework focuses on application-level measures to protect visual data, but will require a static- or dynamic-analysis in order to enforce the design policies at each stage of the application lifetime. We will address this in our future work.

3 BACKGROUND

3.1 Android Permission System Overview

Each application runs in a sandbox with a unique process identity. The Android security model attempts to protect user privacy by requesting and granting permissions at the system level, whether within an application sandbox or between different sandboxes. Within the sandbox, an application is required to request permissions before it can access potentially sensitive information [1]. For instance, an approval request will prompt to the user when the application requests access to the camera, the file system, or the network; all sources of potentially sensitive data. Outside the sandbox, permissions can be customized and checked if the application needs to interact with other applications. Similarly, no data access between applications will be granted if the permission request is declined.

Process-level security enforcement has its limitations. As each application runs as a separate process, all resources within the application are shared and can be accessed as a single shared portion of memory by different parts of the application. As shown in Figure 2a, developers can collect frames easily through network access.

Application	Threat
Augmented markers	Send frames when markers are detected
Augmented faces	Send frames when faces are detected
Text capture	Send text when texts are captured

Table 1: Three threat demonstration applications we studied to understand user privacy concerns in continuous vision computing environments.

Shared resources enable a simplified development process while maintaining the fundamental benefits of the application sandboxing model. Under the current Android permission system, once the user has granted the access to a permission the application has access to all of the capabilities the permission enables, without any limitation. Permission is not requested again through the lifetime of the application install unless the user manually revokes it via system settings. For MR/AR applications in particular, camera data often contains sensitive information such as a person's face or objects in surroundings. Exposure to sensitive information increases with use of the application; thus, an application with continuous access to the camera is potentially subjected to many instances of sensitive information each day. An untrusted application can with relative ease aggregate the data and secretly send them over the network for further inference. The current process-level protections on Android are incapable of protecting the user against this type of data collection.

3.2 Threats in MR/AR Applications

We classify any application that collects camera data as a threat to both user privacy and user security because the information obtained from the camera is always classified as potentially sensitive. In modern mobile systems, any collection of camera information – incidental or malicious – is in the best case a privacy breach and in the worst a security breach. The unrestricted access to sensitive information that applications can obtain in this manner is contrary to the standard mobile computing model designed around the principle of least privilege [1]. In addition, users are not aware of data transferred between the device and the network. The Android permissions model currently presents a single permission dialog to the user when camera access is requested. This gives the application access to capture and record camera information even when the application is in the background. Requesting permission for internet access is declared in the application manifest file, but is automatically granted to the user at install time. With these two permissions granted, an application is able to present a perceived experience to users, perhaps providing them with reasons to accept the permission to access the camera, and without user knowledge transfer camera information over the network.

Our threat demonstration applications shwon in Table 1 use the ARCore framework to present common vision use cases to the user via augmented markers, augmented faces, and text capture, given a situation in which camera information is continuously captured. On each of the three applications, the user is presented with a permission confirmation for the camera the first time the application is opened. In the augmented scenarios, the application shows the user a 3D object rendered over the real world. In the text capture application, any text detected in the camera view is digitized for further use. Each of these applications contain application code, however, that aggregates camera frames that are likely to contain sensitive information. Each time a face or marker is detected with the augmented faces and augmented markers applications, a photo of the face or marker is sent over the network to a private server. In the text capture application, all captured text is streamed to a private server. We are able to demonstrate that these applications are vulnerable to context-sensitive data collection, as shown in Figure 1 – where the credit card information is collected in the augmented marker application.

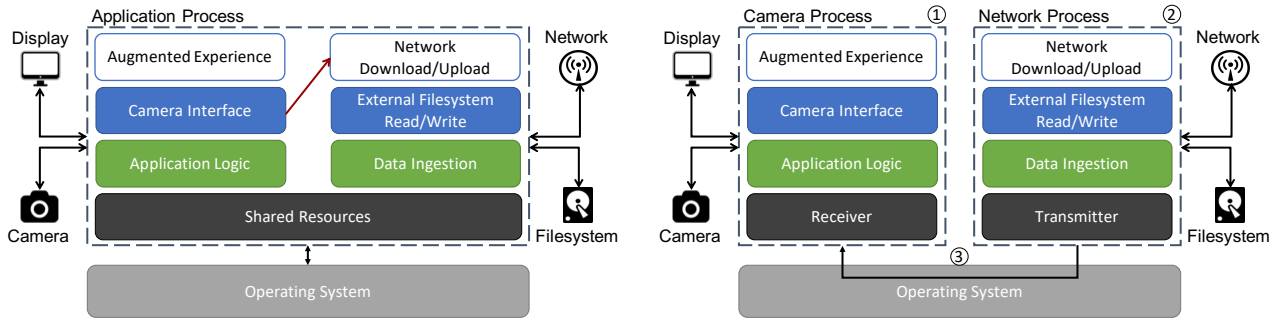
4 DESIGN & IMPLEMENTATION

Our proposed framework is designed around the principles of **resource isolation** and **unidirectional data flow**. We implement our framework on the Android operating system as an application library written in Java and Kotlin.

4.1 Resource Isolation

We require developers to separate the part of the application that requires network access from the rest of the application. Each part of the application runs in its own distinct process. Moving network access into a separate process from camera access provides an added layer of protection for sensitive information in AR and MR apps. We call these two processes the **camera process** and the **network process**, as shown in Figure 2b. The camera process manages the main AR/MR experience, and is denied permission to access the network. It captures camera frames, operates on them, and renders the experience to the user. The camera process also manages the user interface and input of the application. Network communication is accomplished via the network process and information is then transmitted to the camera process. Information transferred may be a downloaded user profile or new 3D models for example. The external filesystem is also a vulnerability because the camera process can write sensitive information to the filesystem and the network process can read from it; we mitigate this by restricting external filesystem access to only the network process. Separation of resources introduces the need for inter-process communication (IPC), which can be computationally expensive, but we found that a typical AR/MR application requires minimal communication in this manner in order to remain performant and maintain a real-time vision-based experience.

4.1.1 Implementation. We implement a Receiver module for the camera process and a Transmitter module for the network process. The Receiver contains the permission definition specific to the camera process, a data-receiver registration API, and a predefined inter-process capable service to handle incoming messages. The transmitter module includes the permission definition specific to the network process, a data-sending API, and a package-local foreground service to send messages in the background. Each library module conducts runtime permissions queries via the system PackageManager before conducting IPC, blocking the developer from breaking framework permission requirements via runtime permission requests or compile-time permission additions. If a permission query detects a policy breach, a permission configuration



(a) In a traditional application, resources are shared in one application process. Thus, developers are able to collect camera frames easily through network access.

(b) In our framework, the camera process cannot access the network ①, the network process cannot access frames ②, and information flow is unidirectional ③.

Figure 2: Our proposed framework augments traditional visual information protection by separating the application into a camera process and a network process with three information flow policies introduced ① ② ③.

exception is thrown by the framework and the IPC is cancelled. A more robust solution is discussed in section §5.3.

4.2 Unidirectional Data Flow

Our framework enforces a unidirectional flow of data from the network process to the camera process. A unidirectional data flow policy of this type gives the sending process – the network process – freedom to communicate arbitrarily large data to the receiving process – the camera process – while trusting the operating system to handle the potential for backchannel leaks. Unidirectional data flow is achieved via platform-supported inter-process communication. Because the system is a trusted part of the threat model, any backchannel communication such as acknowledge signals coming from the receiver are considered contained and therefore benign. With unidirectional data flow in place, the camera process is forbidden from uploading to the network and the network process is forbidden from obtaining sensitive visual information.

4.2.1 Implementation. Our proposed framework uses a bound service to achieve policy-defined unidirectional data flow. The Android Binder framework, an IPC mechanism at the system level, supports inter-process message passing and remote procedure calls. We define a service ReceiverService in the Receiver module that exposes an API to allow remote processes to pass data into it. ReceiverService is then able to transfer that information to the Receiver object and then back to the application's registered receivers. We also define TransmitterService, a service used to initiate network-to-camera transactions in the background. The network application binds locally to TransmitterService via the Transmitter object and initiates a transaction. The transaction process is managed by the framework and the developer is not required to understand the implementation details to utilize the framework API.

4.3 Policy Enforcement

In order for our framework to function, three core policies are introduced. First, the camera process must not have access to make network calls. Second, the network process must not have access to the camera frame or anything inferred from it. Third, the flow of information between the two processes must only be from network to camera and never the other way. Each policy must be enforced on top of the camera process and the network process, as shown in Figure 2b.

4.3.1 Implementation. We apply the existing operating system permissions model to isolate the camera process and the network process by granting proper permissions to the camera process and the network process, relying on the operating system to conduct permission enforcement. In other word, the camera process is restricted from network access, and the network process is restricted from the camera. An application developed under our framework policies that attempts to bypass permissions restrictions will not be able to transact data from the network process to the camera process which is the main application. In addition, the camera process will not be able to bind to the TransmitterService, because it is not externally available. In order to meet our framework's requirements, the developer must separate code to be run in the camera process from code to be run in the network process. We use a platform-enforced method of inter-process communication to guarantee the the flow of information is only from network to camera. The trusted operating system acts as a mediator between the two processes and backchannel communication is limited by system constraints. We prevent the similar types of communication in the opposite direction by hiding the network process transmitter from the camera process, isolating the network process from bound execution, and predefining the transaction API between processes. The camera process is installed without the metadata required to identify the network process. Therefore, the network process is effectively hidden from being accessed externally. We remove the availability of service binding from the network process so that other processes are not able to trigger it. We also define a simple transaction API between services to enforce these requirements for the application developer. Our implementation explores how policy enforcement might be carried out at the application-level. System-level integration is discussed further in section §5.3.

4.4 Limitations/Challenges

With unidirectional information flow enabled and enforced, there are several limitations from the strict resource isolation. For example, in this model, visually-triggered network access is strictly prohibited. Designs should explore the potential for adding visual elements that trigger network events, such as an augmented reality download button. Furthermore, a unidirectional design limits users to engage in multi-user experiences, which also require upstream network access. We hope further research in the field delves deeper into providing more meaningful solutions.

5 EVALUATION

We evaluate our proposed framework on a Samsung Galaxy S8 phone running Android 8.0.0. We run the same augmented reality applications as we discussed in the background (§3) – an augmented markers application, an augmented faces application, and a text capture application. These applications are vulnerable against malicious data collection. The demo applications are implemented using the ARCore platform, but our framework is designed with flexibility in mind, and can be used with other native MR/AR platform tools.

We measure *network traffic* to demonstrate isolation efficiency and *frame rate* to show the potential overhead of the framework. Network traffic is measured according to the Android Studio profiler which provides a plot of network traffic over time for the selected package. Frame rate is measured according to the frame delta reported by ARCore's `Scene.OnUpdateListener` callback. Our analysis requests a frame delta for each frame at a microsecond granularity and writes it to an internal cache file. The final frame-rate is averaged across more than 1000 samples.

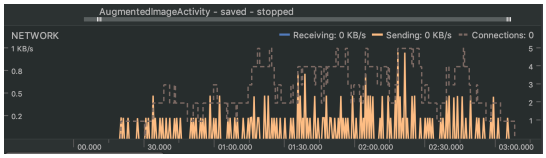
5.1 Results

Preventing data collection from malicious application developers. Network usage is tracked over an application session three minutes long via the Android Studio profiler. Results in Figure 3a shows that a traditional application is able to regularly transmit data, in this case at an average of 1 kilobyte per second. When our implementation is integrated, however, results in Figure 3b shows that application's network usage remains at zero for the entire usage period of the application. Our framework effectively prevents data collection from malicious application developers

Performance overhead. Over an application usage period of 1000 frames, we report the average frame rate in milliseconds. In detail, results in Figure 4 show that our framework only causes a 0.27% (from 33.38 ms to 33.47 ms), 0.68% (from 36.70 ms to 36.95 ms), and 0.03% (from 41.26 ms to 41.27 ms) increase in frame rate, accordingly, in the augmented markers application, the augmented faces application, and the text capture application. It is clear that the required receiver and transmitter services and the associated IPC in our proposed framework slightly decrease the frame rate but the application remains performant. The change in frame rate is not significant enough to merit a noticeably different user experience. Our framework does not incur noticeable performance overhead.

5.2 Impact

We introduce a framework design that can protect private user visual data in MR/AR applications at a low cost. An end-to-end implementation of the suggested design has the potential to protect billions of users' visual information, as continuous MR/AR experiences are becoming more and more popular. Our proposed framework will involve building a trusted platform and necessary tools for application developers to work with. We anticipate that our work will build trust between MR/AR application users and MR/AR application developers. In addition, our proposed framework opens doors for new research into not only the limited types of applications discussed, but for applications which require complex application-network interaction, such as multi-user experiences or applications which tightly couple user interface elements and network activity.



(a) The marker detection threat demonstration application generates consistent network traffic.

(b) The same application generates zero network traffic, confined by our proposed framework.

Figure 3: Our framework effectively prevents malicious network access, i.e., no frame is collected.

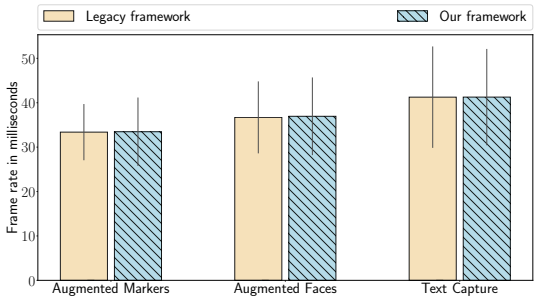


Figure 4: Frame rate is similar across all tested applications, with and without framework integration.

5.3 Future work

Operating system integration and developer tools We have demonstrated that resource isolation can protect users against malicious data collection. However, the current implementation is realized by modifying code at the application level. To ease the burden of the developers, we plan to integrate the introduced framework into the Android OS. This will provide developer tools to enhance the process, including IDE integrations with static analysis feedback, a public API to interact with the system-level framework from

within an application, and dynamic information flow analysis to validate policy enforcement.

Framework certification model In the current implementation, we do not address how to strictly enforce the proposed policies in the Android OS, but instead trust the system to manage the majority of policy enforcement. To verify that applications are following our proposed policies, we will explore choices for implementing permission enforcement at application install-time and notify the user whether the requested application observes the framework's requirements.

6 CONCLUSION

As MR/AR technology matures, continuous visual information will allow new application categories to develop but with rising privacy concerns. Providing frameworks like the one presented in this work will allow users protection against malicious applications that collect sensitive information. Sensitive information is best protected by restricting its use to the device, as once it is available to the network a high level of trust is required between the user and the application developer. User privacy will drive the growth of MR/AR as practical technologies; as users become more confident in the protection of their information, they are more likely to adopt new technology.

REFERENCES

[1] Android Developers. https://developer.android.com.

[2] Sven Bugiel Abdallah Dawoud. 2019. DroidCap: OS Support for Capability-based Permissions in Android. In *NDSS*.

[3] Alessandro Acquisti, Ralph Gross, and Fred Stutzman. 2011. Privacy in the Age of Augmented Reality.

[4] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-Pic: A Platform for Privacy-Compliant Image Capture. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. ACM, New York, NY, USA, 235–248. https://doi.org/10.1145/2906388.2906412

[5] Stefano Baldassi, Tadayoshi Kohno, Franziska Roesner, and Moqian Tian. 2018. Challenges and New Directions in Augmented Reality, Computer Security, and Neuroscience - Part 1: Risks to Sensation and Perception. *CoRR* abs/1806.10557 (2018).

[6] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. 2011. Analyzing Inter-application Communication in Android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. ACM, New York, NY, USA, 239–252. https://doi.org/10.1145/1999995.2000018

[7] Josh Constine. 2018. Snapchat launches Spectacles V2, camera glasses you'll actually wear. https://techcrunch.com/2018/04/26/snapchat-spectacles-2/. (2018).

[8] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. 2005. Labels and Event Processes in the Asbestos Operating System. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05)*. ACM, New York, NY, USA, 17–30. https://doi.org/10.1145/1095810.1095813

[9] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 393–407. http://dl.acm.org/citation.cfm?id=1924943.1924971

[10] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, Berkeley, CA, USA, 531–548. http://dl.acm.org/citation.cfm?id=3241094.3241136

[11] Google. 2019. ARcore. https://developers.google.com/ar//. (2019).

[12] S. Jana, A. Narayanan, and V. Shmatikov. 2013. A Scanner Darkly: Protecting User Privacy from Perceptual Applications. In *2013 IEEE Symposium on Security and Privacy*. 349–363. https://doi.org/10.1109/SP.2013.31

[13] Limin Jia, Jassim Aljuraidan, Elli Fragkaki, Lujo Bauer, Michael Stroucken, Kazuhide Fukushima, Shinsaku Kiyomoto, and Yutaka Miyake. 2013. Run-Time Enforcement of Information-Flow Properties on Android. In *Computer Security – ESORICS 2013*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 775–792.

[14] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information Flow Control for Standard OS Abstractions. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. ACM, New York, NY, USA, 321–334. https://doi.org/10.1145/1294261.1294293

[15] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. 2018. Arya: Operating System Support for Securely Augmenting Reality. *IEEE Security Privacy* 16, 1 (January 2018), 44–53. https://doi.org/10.1109/MSP.2018.1331020

[16] Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. 2018. Towards Security and Privacy for Multi-user Augmented Reality: Foundations with End Users. *2018 IEEE Symposium on Security and Privacy (SP)* (2018), 392–408.

[17] S. M. Lehman and C. C. Tan. 2017. PrivacyManager: An access control framework for mobile augmented reality applications. In *2017 IEEE Conference on Communications and Network Security (CNS)*. 1–9. https://doi.org/10.1109/CNS.2017.8228630

[18] Microsoft. 2019. HoloLens2. https://www.microsoft.com/en-us/hololens/apps. (2019).

[19] H. Okhravi, S. Bak, and S. T. King. 2010. Design, implementation and evaluation of covert channel attacks. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. 481–487. https://doi.org/10.1109/THS.2010.5654967

[20] Nisarg Raval, Animesh Srivastava, Kiron Lebeck, Landon Cox, and Ashwin Machanavajjhala. 2014. MarkIt: Privacy Markers for Protecting Visual Secrets. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 1289–1295. https://doi.org/10.1145/2638728.2641707

[21] Franziska Roesner, David Molnar, Alexander Moshchuk, Tadayoshi Kohno, and Helen J. Wang. 2014. World-Driven Access Control for Continuous Sensing. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1169–1181. https://doi.org/10.1145/2660267.2660319

[22] Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. McKinley, and Emmett Witchel. 2009. Laminar: Practical Fine-grained Decentralized Information Flow Control. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09)*. ACM, New York, NY, USA, 63–74. https://doi.org/10.1145/1542476.1542484

[23] Güliz Seray Tuncay, Soteris Demetriou, Karan Ganju, and Carl A. Gunter. 2018. Resolving the Predicament of Android Custom Permissions. In *NDSS*.