Protecting Visual Information in Augmented Reality

from Malicious Application Developers

by

Jordan Kyle Jensen

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Robert LiKamWa, Chair
Adam Doupé
Ruoyu Wang

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Visual applications – those that use camera frames as part of the application – provide a rich, context-aware experience. The continued development of mixed and augmented reality (MR/AR) computing environments furthers the richness of this experience by providing applications a continuous vision experience, where visual information continuously provides context for applications and the real world is augmented by the virtual. To understand user privacy concerns in continuous vision computing environments, this work studies three MR/AR applications (augmented markers, augmented faces, and text capture) to show that in a modern mobile system, the typical user is exposed to potential mass collection of sensitive information, posing privacy and security deficiencies to be addressed in future systems.

To address such deficiencies, a development framework is proposed that provides resource isolation between user information contained in camera frames and application access to the network. The design is implemented using existing system utilities as a proof of concept on the Android operating system and demonstrates its viability with a modern state-of-the-art augmented reality library and several augmented reality applications. Evaluation is conducted on the design on a Samsung Galaxy S8 phone by comparing the applications from the case study with modified versions which better protect user privacy. Early results show that the new design efficiently protects users against data collection in MR/AR applications with less than 0.7% performance overhead.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Robert LiKamWa for his support and mentorship through the duration of my study at ASU. He shows belief in me when I struggle, and he balances his effort to push me to grow with his desire to help me learn. I take pride in being a member of Meteor Studio and I look forward to watching its growth in coming months and years. I appreciate the close mentorship and assistance of Jinhan Hu through this work. He guided my work and encouraged me to give my best. The other members of Meteor Studio each played a part in this work, and I want to thank each one of them. I also want to thank professors Adam Doupé and Fish Wang for being willing to contribute to this work by reviewing and providing feedback.

I am grateful for Dr. Junshan Zhang and the Information Networks Group, who allowed me to join their work early on in my program and provided support as I transitioned to Meteor Studio. The Android skills I gained there allowed me to think deeply about AR privacy and its implications.

Perhaps most influential in this work was my wife, who worked tirelessly to provide a good life for us while I pursued a higher degree. She has never failed to bolster my confidence, believe in myself, and move forward despite trials. Additionally, my parents and siblings have been ever supportive in my aspirations to do good.

The University deserves my utmost gratitude for the combined effort to make Arizona State such a beautiful and welcoming place. The School of Computing, Informatics, and Decision Systems Engineering and the School of Electrical, Computer, and Energy Engineering have provided an academically challenging growth experience I will carry with me for the rest of my career.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

As mobile computing evolves, its applications shift to include a broader and deeper level of functionality. The growth of visual perception applications on mobile demonstrates that with such a shift come new application types and new vulnerabilities to user privacy and security. Visual applications 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. Meanwhile, visual applications raise serious security concerns because they often contain private and sensitive user information that can be utilized maliciously by application developers. Even though mobile operating systems such as Android are equipped with permission systems, they still fail to protect user privacy. MR/AR applications are particularly vulnerable because they have unrestricted continuous access to sensitive visual information without transparency as to how it is used Baldassi *et al.* (2018); Lebeck *et al.* (2018); Acquisti *et al.* (2011).

We conduct a case study around three MR/AR applications running on the ARCore framework. The applications are *augmented markers*, which renders 3D objects above the markers (as shown in Figure 1.1); *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 include hidden functionality to send visual information to a local server when a marker, a face, or text is detected accordingly. The threat model we address in this work is demonstrated by these applications. We design for cases in

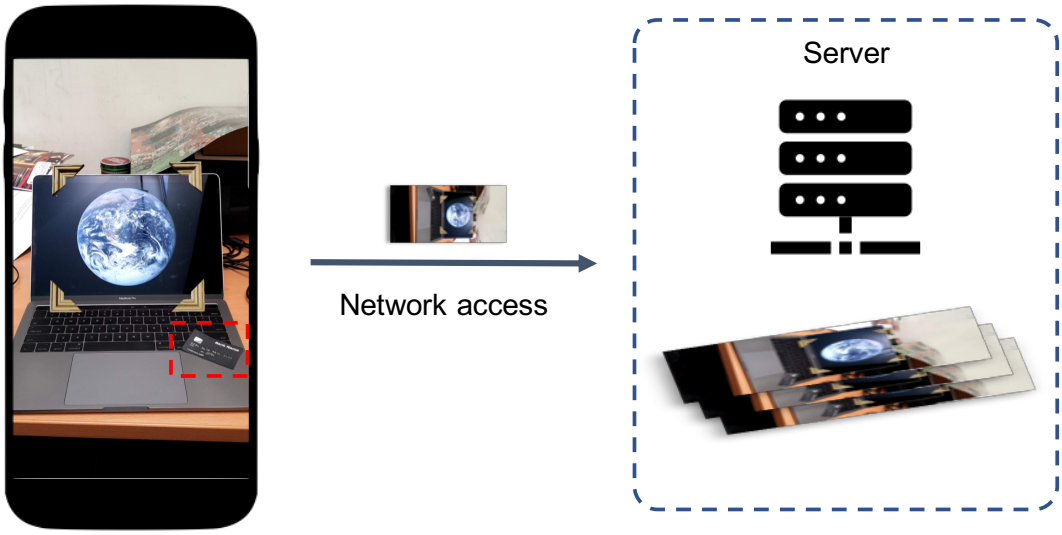


Figure 1.1: Camera frames in MR/AR applications often contain sensitive information (e.g. a credit card) which is vulnerable to collection by the application developer.

which the application developer seeks to 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.

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 Krohn *et al.* (2007); Enck *et al.* (2010); Fernandes *et al.* (2016); Efstathopoulos *et al.* (2005); Roy *et al.* (2009) or adding an intermediate layer to preprocess frames and filter out sensitive visual information Jana *et al.* (2013); Roesner *et al.* (2014); Aditya *et al.* (2016); Raval *et al.* (2014). However, they add unnecessary complexity and performance overhead which limits their practicality for real-world MR/AR applications.

We propose a 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. Our solution is MR/AR-specific, and addresses problems unique to continuous vision applications. We aim to raise barriers against visual information collection by guiding application development and enforcing information flow policies. We assume a trusted system and leverage the Android permissions system to reduce framework overhead. Our proposed framework is designed to protect against visual information leaks at the highest level of granularity with the most visibility – application shared resources. It is for this reason that protection against covert channel attacks is not incorporated into our design. We also focus on developer freedom by not locking the developer into one computer 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 the implementation of our proposed framework on a Samsung Galaxy S8 phone running modified versions of the applications in our case study. By monitoring *network traffic* using the Android Studio profiler and application *frame rate*, early results show that our framework effectively prevents malicious network access with negligible performance overhead (less than 0.7%).

We make the following contributions.

- We identify that MR/AR applications running on top of 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 proposed framework can protect MR/AR application users against data collection in real-time.

Chapter 2

RELATED WORK

Information Flow Control Krohn et al. introduced the Flume system to allow safe interaction between conventional and DIFC-aware processes Krohn *et al.* (2007). Enck et al. presented TaintDroid to identify and protect data leakage via untrusted applications Enck *et al.* (2010). Fernandes et al. introduced FlowFence to guarantee that sensitive data is only processed within designated functions that run in FlowFence-provided sandboxes Fernandes *et al.* (2016). Roy et al. presented Laminar which implements and further optimizes DIFC for program objects and OS resources Roy *et al.* (2009). Efstathopoulos et al. used Asbestos labels to isolate user data for information flow control Efstathopoulos *et al.* (2005). However, each of these generalized approaches trade 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 Jana *et al.* (2013). Roesner et al. introduced a framework that has a granularity to manage objects in the sensing streams Roesner *et al.* (2014). Lebeck et al. introduced Arya system to secure the rendering of AR applications Lebeck *et al.* (2018). Aditya et al. presented the I-Pic platform for policy-compliant image capture Aditya *et al.* (2016). Lehman et al. developed PrivacyManager to help developers control malicious functionalities in AR applications Lehman and Tan (2017). Raval et al. proposed the MarkIt framework to allow

users to specify and enforce fine-grained control over video feeds Raval *et al.* (2014). 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 Jia *et al.* (2013). Tuncay et al. introduced Cusper to allow applications to change permissions dynamically Tuncay *et al.* (2018). ComDroid analyzes intent statically Chin *et al.* (2011). Laminar enforces the security policies at runtime Roy *et al.* (2009). Each of these works explore modifications to policy management, but do not for real-time applications. The design of our proposed 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. Building upon policy enforcement will ensure that side-channel information leaks are minimized in addition to explicit information leaks Our implementation accomplishes information flow control at a rudimentary level by enforcing a unidirectional data flow via an application-level service and binding mechanism.

Chapter 3

BACKGROUND

3.1 A case study

Privacy protection in MR/AR has been actively discussed mainly from five aspects, including input protection, data protection, output protection, user interactions protection, and device protection de Guzman *et al.* (2018). In this section, we focus on presenting the limitations and problems in current mobile operating systems for preventing **data collection within an MR/AR application** – by studying the most salient vulnerabilities and designing an application which exploits them.

3.1.1 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 android.permissions (2019). 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. 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 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, as shown in Figure 4.1a.

3.1.2 Threats in MR/AR Applications

Our threat demonstration applications 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 1.1 – where credit card information is collected in the augmented marker application. 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 android.leastprivilege (2019). 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 the user, perhaps providing them with reason to accept permission to access the camera, and without user knowledge transfer camera information over the network.

Chapter 4

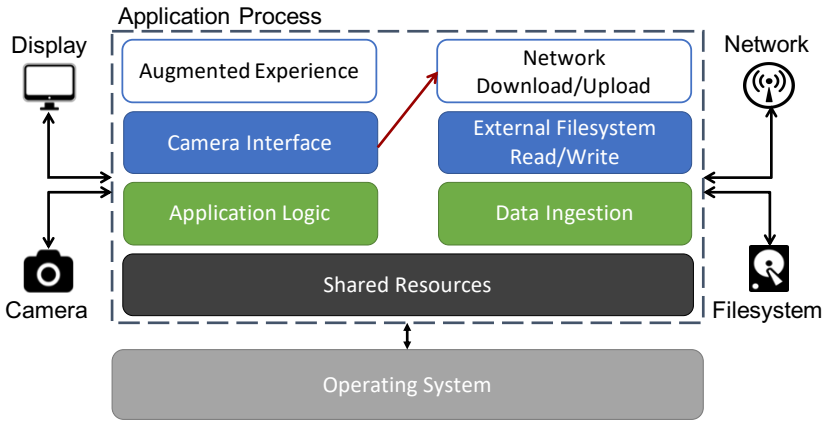
DESIGN

In this section, we propose an application-level framework that gives the developer implementation freedom by allowing visual computing to be done in any manner, while imposing on the developer a development architecture that separates visual computing from the network. The goal of the framework is to protect users against malicious collection of sensitive visual data by isolating network access from the rest of the application and limiting network communication without impeding the real-time nature of augmented experiences, affecting performance severely, or requiring significant adjustment from the developer to use.

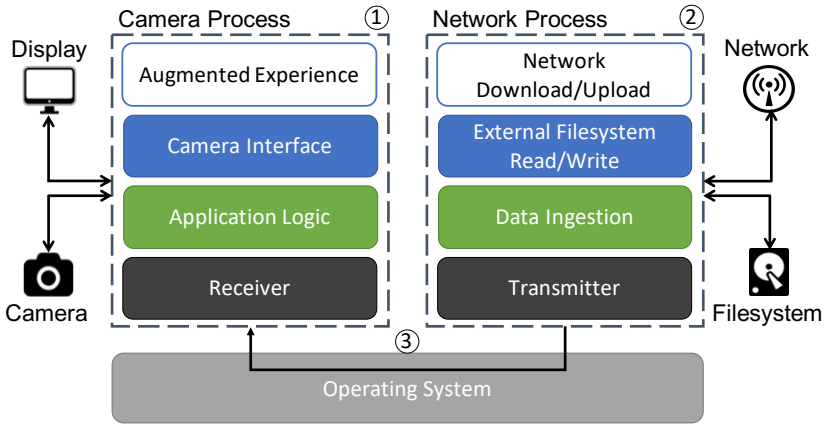
4.0.1 Overview

Our proposed framework is designed around the principles of resource isolation and unidirectional data flow, as shown in Figure 4.1b.

First, we separate the part of the application that requires network access from the rest of the application into its own distinct process. Resource isolation in this way restricts resource access – particularly that of the camera and the network – and buffers application network traffic. We then deny network permissions to the main application process as a check that all network processing is only conducted inside the network process. Similarly, the network process is denied permission to use the device camera to decouple network access from camera access. Separation into two processes is chosen as opposed to other isolation methods because with it we are able to utilize existing operating system permissions constructs. Implementing custom permissions enforcement on top of the system would add unnecessary complexity to the design, as



(a) In a traditional application, resources are shared within one application process. The application developer is able to collect camera frames easily through direct network access.



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

Figure 4.1: 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.

standard process separation fulfills the basic requirements of visual resource isolation.

Second, to guarantee that necessary information flow is still available from the main application to the network process, we further isolate the network by adding a unidirectional data flow policy which prevents the application from communicating sensitive visual information to the network process directly. Isolating computing resources and enforcing unidirectional data flow preempts a developer's attempts to collect visual information by providing significant barriers between the camera and the network. The main application thus is left without an obvious approach to transmitting user data over the network.

However, even with unidirectional information flow enabled, there are some limitations in strict resource isolation. For example, user-triggered upstream network access is not covered by this work except in the case of system notifications discussed in section 5.0.1. We did not explore deeply the potential for adding visual elements that trigger network events, such as an augmented reality download button. Furthermore, our design does not protect users in multi-user experiences, which also require upstream network access. In these cases we make the assumption that the developer may use the traditional application architecture, provided that the user is notified of the potential threat this poses to their security and privacy. We hope further research in the field delves deeper into providing meaningful solutions in these cases in addition to those covered by our work

4.0.2 Resource Isolation

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**. 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.

Isolating network access from visual computation prevents the developer from sending a camera frame (or data extracted from it) that is available in memory explicitly to an external server. 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. Our framework is designed under the assumptions that network access is infrequent and that while a mixed-reality experience is being shown to the user, network access is not tightly coupled to rendering time. However, with isolation in place the camera process is still able to transmit visual information via various IPC methods to the network process. Our framework uses a unidirectional data flow policy to manage unrestricted IPC access.

4.0.3 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.

Our framework accomplishes this 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.

However, a unidirectional approach has limitations when applied to MR/AR. More specifically, enforcing unidirectional data flow from the network may limit applications that require visual computation offloading or bidirectional network access. Computation offloading is used in order to preserve battery life and compartmentalize an application's components, both of which are concerns in MR/AR applications, but offloading visual computation introduces a flow of data that breaks framework policies. In addition, multi-user augmented reality experiences are growing in popularity but require bidirectional network access. Our work does not account for the cases in which the app requires bidirectional access to the network and instead leaves them for future work.

4.0.4 Policy Enforcement

In order for our framework to function, three core policies are introduced. Each policy must be enforced on top of the camera process and the network process, as shown in Figure 4.1b.

First, the camera process must not have access to make network calls. We apply the existing operating system permissions model to handle this policy by granting the camera permission to and revoking the network permission from the camera process, relying on the operating system to conduct permission enforcement. In order to meet

framework requirements, the developer must separate code to be run in the main (camera) process from code to be run in the network process.

Second, the network process must not have access to the camera frame or anything inferred from it. The same principles apply here as with the camera process. We grant the network permission and revoke the camera permission and let the operating system handle enforcement.

Third, the flow of information between the two processes must only be from network to camera and never the other way. It is for this purpose that we use a platform-enforced method of inter-process communication. 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.

Chapter 5

IMPLEMENTATION

In this section, we present the initial implementation of our proposed framework on the Android operating system as an application library written in Java and Kotlin. The library contains two modules, Receiver for the camera process and Transmitter for the network process. The receiver module 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. We also propose initial plans to implement framework policy enforcement on future handheld and head-mounted mobile devices. Our implementation augments the Android security model by leveraging existing application-level constructs to provide the protections discussed while maintaining developer flexibility and application performance.

5.0.1 Resource Isolation

Resource isolation on a mobile Android device requires unique permissions for the camera process and the network process. Android permissions are assigned according to the application package, so in our implementation the two application processes are separated into two corresponding application packages. The framework's viability is entirely dependent on developer adoption of the framework. Our approach makes the assumption that developer adoption is enforced, whether at development-time, compile-time, install time, or during application store review.

We create two individual application modules in the same Android Studio project, and designate one as the network package. Each of the library modules are associated with the corresponding application package module.

The Receiver library module is compiled with the camera package, which contains the main application code, including the software camera pipeline, and the AR experience. The main application initializes the Receiver object via a single line of code:

```
val receiver = Receiver(applicationContext)
```

An object representing the framework interface is instantiated with the current application context, associating it with the running application package. The application then registers a receiver with the desired string identifier and the data to be sent as a byte array:

```
receiver.registerReceiver(("msg_identifier", bytes) -> {
  // Load data into destination
})
```

The Transmitter module is compiled with the network package, which is limited to developer-defined network code. Network application code is required to initialize the Transmitter object in a similar manner to that of the receiver:

```
val transmitter = Transmitter(applicationContext)
```

The Transmitter "send" method is then called with a string identifier and a byte array containing the data when the network application needs to send data to the camera application:

```
transmitter.send("msg_identifier", bytes)
```

Android application permissions are typically defined by the developer in the AndroidManifest.xml application metadata file. We use this method in our implementation, including a manifest with each framework library module that is merged with the application manifest. 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 PermissionConfigurationException is thrown by the framework and the IPC is cancelled. A more robust solution is discussed in the future work section.

The addition of user-triggered network events is accomplished via system notifications. When the network package is installed, the application may present a set of notifications that broadcast local messages to perform various network actions. Since the synchronicity of network traffic may not always be defined at install-time, asynchronously-triggered network events introduce a wider set of application support, such as that of the face overlay 3D model transfer in the augmented faces application with framework integration. In this way, the network package is able to remain isolated from the camera package and offer asynchronous support.

5.0.2 Unidirectional Data Flow

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.

5.0.3 Policy Enforcement

Our proposed framework requires that the network be isolated from the main application, the camera be isolated from the network application, and that information flow be unidirectional from the network application to the camera application. Our implementation explores how policy enforcement might be carried out at the application-level. System-level integration is discussed further in the future work section. The combination of resource isolation via separate application packages and unidirectional data flow via bound services in our initial implementation demonstrates that with minor adjustments to application development users may be protected from collection of visual data.

The main application is restricted from network access, and the network application 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 application to the main application. In addition, the camera application will not be able to bind to the TransmitterService, because it is not externally available.

Information flow is controlled from network to camera, and backchannel leaks are minimized. The network application can bind to the camera application to initiate a transaction, and the trusted operating system manages backchannel signals coming from the camera application.

True policy enforcement will require further system integration. With the frame-

work in place, the developer is still free to use alternative methods of inter-process communication. Local sockets, custom bound services, and broadcast receivers remain available to the developer to leak visual information to the network package. Because these are system-level constructs, our implementation treats them as trusted. However, in an operating system with our proposed framework enforcement built in, the network process would be better protected from external access.

Chapter 6

EVALUATION

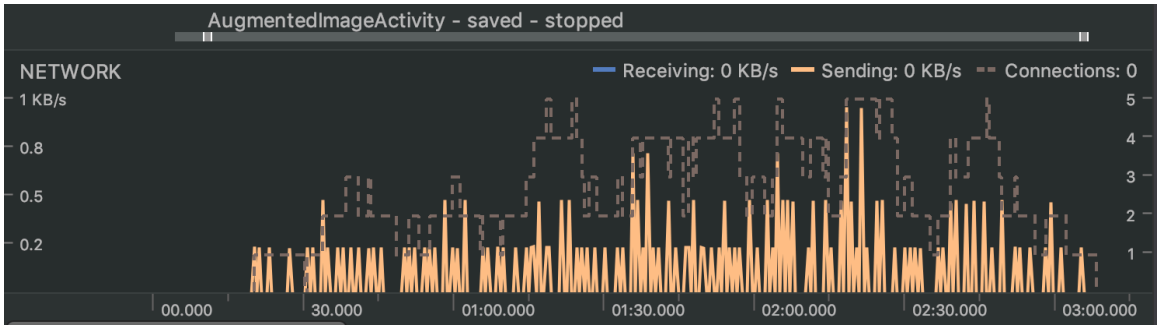
We evaluate our proposed framework on a Samsung Galaxy S8 phone running Android 8.0.0. We run the same augmented reality applications as we discussed in the case study §3.1, including an augmented markers application, an augmented faces application, and a text capture application. These applications have been demonstrated to be 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. The Android Studio profiler provides a plot of network traffic over time for the selected package and illustrates outgoing and incoming data.

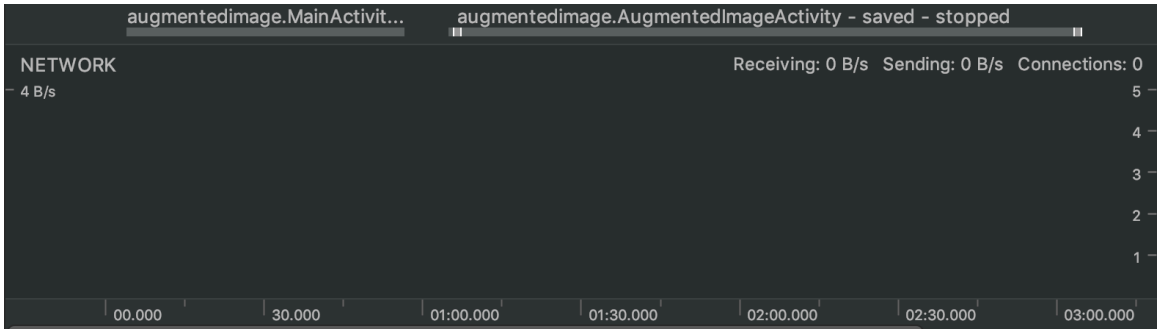
Frame rate is measured according to the frame delta reported by ARCore's `Scene.OnUpdateListener` callback. ARCore provides a simple API for receiving frame time deltas for each frame rendered that were used during each capture session. Our analysis requests a frame delta for each frame at a microsecond granularity and writes it to an internal cache file. The final frame-rate is averaged across more than 1000 samples. In an attempt to prevent overhead from frame measurement itself, we write only the requested frame time delta to the file, and then perform analysis off-device and after the capture session.

6.0.1 Early results

Our evaluation of the three applications reveals the following observations.



(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 6.1: When compared to traditional applications, applications with our framework integrated effectively prevent malicious network access, i.e. no frame is collected.

Our framework effectively prevents malicious data collection, as shown in Figure 6.1. Network usage is tracked over an application session three minutes long via the Android Studio profiler. A traditional application is able to regularly transmit data, in this case at an average of 1 kilobyte per second, as shown in Figure 6.1a. When our implementation is integrated, however, application network usage remains at zero for the entire usage period of the application, as shown in Figure 6.1b.

Our framework does not incur noticeable performance overhead, as shown in Figure 6.2. Over an application usage period of 1000 frames or more, we

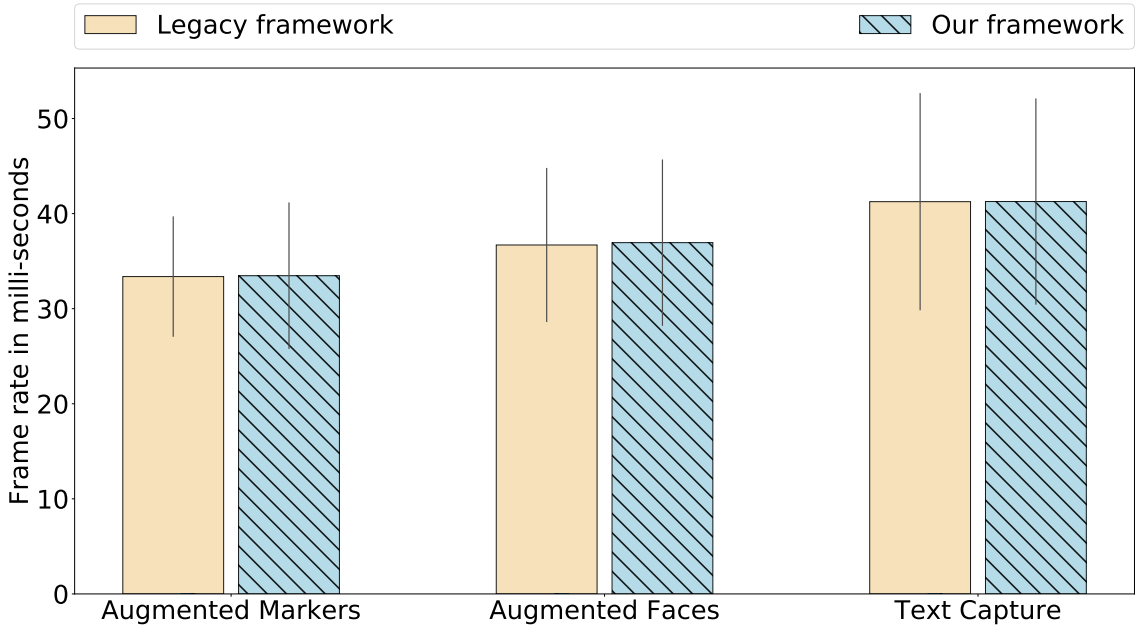


Figure 6.2: Performance (frame rate in milliseconds) is similar across all tested applications, with and without framework integration.

report average frame length in milliseconds. In detail, it 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 length, 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 length is not significant enough to merit a noticeably different user experience.

6.0.2 Impact

We introduce a framework design that can protect sensitive 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.

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. Our work introduces research challenges that necessitate continuing work in understanding software development and data structure patterns in split-process application construction.

6.0.3 Future work

Operating System Integration We have demonstrated the effect the framework resource isolation has on user protection 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 Android at the system level. A deeply integrated version of our framework 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 on the fly.

Framework Certification Model In the current implementation, we do not address how to strictly enforce the proposed policies in an Android environment, but instead trust the system to manage the majority of policy enforcement To verify that applications are following our proposed policies, future work will implement permission enforcement at application install-time and notify the user whether the requested

application observes the framework's requirements. We further intend to prevent typical side-channel IPC attacks via local sockets, custom bound services, and broadcast receivers. Future implementations may include system-level unidirectional policy enforcement similar to the SELinux-based unidirectional data flow described by Shimko et al Shimko and Brindle (2007).

Benchmarks This work serves as a preliminary demonstration to reveal whether that the proposed framework is effective in protecting the MR/AR users against malicious application developers and to drive novel research in the field of MR/AR privacy and security. We did not compare the performance of our framework to other works at this time. We plan to choose other state-of-the-art privacy protection systems as benchmarks to compare against in future work.

Chapter 7

CONCLUSION

As MR/AR technology matures, continuous visual information will allow new application categories to develop. But rising privacy concerns will accompany new application categories. 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. Future work in this area will provide stronger guarantees through lower-level, more extensive policy enforcement. 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

Acquisti, A., R. Gross and F. Stutzman, "Privacy in the age of augmented reality", (2011).

Aditya, P., R. Sen, P. Druschel, S. Joon Oh, R. Benenson, M. Fritz, B. Schiele, B. Bhattacharjee and T. T. Wu, "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, pp. 235–248 (ACM, New York, NY, USA, 2016), URL http://doi.acm.org/10.1145/2906388.2906412.

android.leastprivilege, "Android developers — application fundamentals. https://developer.android.com/guide/components/fundamentals", URL https://developer.android.com/guide/components/fundamentals (2019).

android.permissions, "Android developers — request app permissions. https://developer.android.com/training/permissions/requesting", URL https://developer.android.com/training/permissions/requesting (2019).

Baldassi, S., T. Kohno, F. Roesner and M. Tian, "Challenges and new directions in augmented reality, computer security, and neuroscience - part 1: Risks to sensation and perception", CoRR **abs/1806.10557** (2018).

Chin, E., A. P. Felt, K. Greenwood and D. Wagner, "Analyzing inter-application communication in android", in "Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services", MobiSys '11, pp. 239–252 (ACM, New York, NY, USA, 2011), URL http://doi.acm.org/10.1145/1999995.2000018.

de Guzman, J. A., K. Thilakarathna and A. Seneviratne, "Security and privacy approaches in mixed reality: A literature survey", CoRR **abs/1802.05797**, URL http://arxiv.org/abs/1802.05797 (2018).

Efstathopoulos, P., M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek and R. Morris, "Labels and event processes in the asbestos operating system", in "Proceedings of the Twentieth ACM Symposium on Operating Systems Principles", SOSP '05, pp. 17–30 (ACM, New York, NY, USA, 2005), URL http://doi.acm.org/10.1145/1095810.1095813.

Enck, W., P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel and A. N. Sheth, "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, pp. 393–407 (USENIX Association, Berkeley, CA, USA, 2010), URL http://dl.acm.org/citation.cfm?id=1924943.1924971.

Fernandes, E., J. Paupore, A. Rahmati, D. Simionato, M. Conti and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks", in "Proceedings of the 25th USENIX Conference on Security Symposium", SEC'16, pp. 531–548 (USENIX Association, Berkeley, CA, USA, 2016), URL http://dl.acm.org/citation.cfm?id=3241094.3241136.

Jana, S., A. Narayanan and V. Shmatikov, "A scanner darkly: Protecting user privacy from perceptual applications", in "2013 IEEE Symposium on Security and Privacy", pp. 349–363 (2013).

Jia, L., J. Aljuraidan, E. Fragkaki, L. Bauer, M. Stroucken, K. Fukushima, S. Kiyomoto and Y. Miyake, "Run-time enforcement of information-flow properties on android", in "Computer Security – ESORICS 2013", edited by J. Crampton, S. Jajodia and K. Mayes, pp. 775–792 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013).

Krohn, M., A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler and R. Morris, "Information flow control for standard os abstractions", in "Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles", SOSP '07, pp. 321–334 (ACM, New York, NY, USA, 2007), URL http://doi.acm.org/10.1145/1294261.1294293.

Lebeck, K., K. Ruth, T. Kohno and F. Roesner, "Arya: Operating system support for securely augmenting reality", IEEE Security Privacy **16**, 1, 44–53 (2018).

Lebeck, K., K. Ruth, T. Kohno and F. Roesner, "Towards security and privacy for multi-user augmented reality: Foundations with end users", 2018 IEEE Symposium on Security and Privacy (SP) pp. 392–408 (2018).

Lehman, S. M. and C. C. Tan, "Privacymanager: An access control framework for mobile augmented reality applications", in "2017 IEEE Conference on Communications and Network Security (CNS)", pp. 1–9 (2017).

Raval, N., A. Srivastava, K. Lebeck, L. Cox and A. Machanavajjhala, "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, pp. 1289–1295 (ACM, New York, NY, USA, 2014), URL http://doi.acm.org/10.1145/2638728.2641707.

Roesner, F., D. Molnar, A. Moshchuk, T. Kohno and H. J. Wang, "World-driven access control for continuous sensing", in "Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security", CCS '14, pp. 1169–1181 (ACM, New York, NY, USA, 2014), URL http://doi.acm.org/10.1145/2660267.2660319.

Roy, I., D. E. Porter, M. D. Bond, K. S. McKinley and E. Witchel, "Laminar: Practical fine-grained decentralized information flow control", in "Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation", PLDI '09, pp. 63–74 (ACM, New York, NY, USA, 2009), URL http://doi.acm.org/10.1145/1542476.1542484.

Shimko, S. and J. Brindle, "Securing inter-process communications in selinux", 2007 SELinux Symposium (2007).

Tuncay, G. S., S. Demetriou, K. Ganju and C. A. Gunter, "Resolving the predicament of android custom permissions", in "NDSS", (2018).