Arya: Operating System Support for Securely Augmenting Reality

Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner | University of Washington

Augmented reality (AR) applications capture sensor input from a user's surroundings and overlay virtual output on their perception of the world, enabling new, immersive experiences. However, this technology raises serious security and privacy risks such as malicious or buggy AR output.

Augmented reality (AR) technologies enable users to interact with virtual content in fundamentally new ways. AR applications capture input from a user's surroundings, such as video, depth sensor data, or audio, and they overlay output (for instance, visual, audio, or haptic feedback) directly on the user's perception of the real world, through devices like smartphones, head-mounted displays (HMDs), or automotive windshields.

While the vision of AR is decades old,[1] AR technologies are only now on the cusp of commercial viability and beginning to capture the attention of users worldwide. From the wildly popular mobile AR app Pokémon Go to powerful HMDs like Microsoft's HoloLens and Meta's Meta2, as well as AR-enabled car windshields[2] and military applications,[3] interest in AR technologies across diverse industry sectors is increasing. Figure 1 shows two examples.

Although AR technologies promise great potential benefits, they also raise new and serious computer security and privacy risks. For example, AR applications' need for rich, continuous sensor data (such as video and audio feeds) raises privacy concerns for both users and bystanders. The ability for AR applications to generate virtual (visual, audio, or haptic) content that modifies a user's perception of the physical world also raises new security and safety risks. For example, consider a buggy or malicious AR windshield application that obscures real-world pedestrians, overlays misleading information on real-world road signs, startles the user while driving, or strategically obstructs virtual content from another, simultaneously running AR application. Addressing AR output risks is particularly critical for fully immersive AR systems, such as HMDs and car windshields, where users cannot easily disengage from their devices if output security issues arise.

Figure 2 shows an abstract architecture of an AR platform, with sensor input coming in and virtual content produced as output. The academic computer security community has begun turning its attention to the potential input and output risks with AR[4]—focusing primarily on risks from buggy or malicious applications rather than the AR platform itself—and exploring potential solutions to mitigate these risks. On the input side, prior efforts have studied user perception of AR privacy risks[5] and worked to mitigate these risks, for instance, by limiting the amount of sensor data made available to AR applications[6–8] or by enforcing context-based policies on sensor data collection.[9,10] (This and additional related work is discussed further in the conference version of this article.[11]) However, little work has considered risks or mitigations on the output side.

In this article, we thus discuss the potential security risks of AR output from buggy or malicious applications, and we explore how an AR operating system can be designed to mitigate these risks. Specifically, we describe the design of Arya, a prototype AR platform with output security as an explicit, first-class goal. In our threat model, Arya is trusted, but the AR applications running on Arya are untrusted. With Arya's security mechanisms enabled, applications still have significant flexibility to create immersive AR experiences, but their visual content is constrained by the platform based on output policies, such as ensuring that windshield applications cannot obscure real-world road signs or pedestrians while the car is moving. This work, which is described in more detail in our conference and workshop papers,[11,12] both identifies and overcomes numerous challenges in designing AR systems to mitigate output security and safety risks.

We stand today at a pivotal juncture with AR technologies, just as we did in the early 2000s with smartphones—there are clear indicators that these emerging technologies are on the horizon, yet it is still very early in their life cycles. Thus, now is the time to consider security for AR, while the technologies are still young and designs are not yet set in stone.

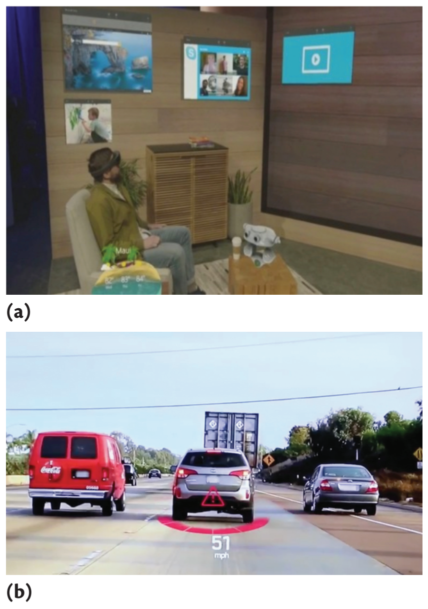

Figure 1. Examples of augmented reality (AR). (a) A Microsoft HoloLens demo showcasing multiple Windows 10 applications (image source: www.geek.com/microsoft/microsofts-hololens-demo-steals-the-show-at-build-2015-1621727/). (b) Hyundai's windshield demo at the Consumer Electronics Show (image source: https://www.youtube.com/watch?v=iZg89ov75QQ). Notice the AR warning sign partially occluding the car.

Motivation and Threat Model

Unlike today's single-app AR experiences (for instance, AR smartphone games like Pokémon Go), we envision that with emerging immersive HMD platforms like HoloLens, users will wish to run multiple AR applications simultaneously augmenting their views of the world. For example, while playing a game like Pokémon Go, users may also wish to use an app that overlays walking directions to nearby restaurants, or that recognizes and identifies nearby social media contacts. To reap the full benefits of these apps, the user must use them while actively moving about and interacting with the real world.

Unfortunately, in addition to creating novel opportunities, AR applications have a unique ability to impact users' perceptions of the real world in undesirable or harmful ways. Specifically, the interaction of multiple AR apps with each other and with the user's view of the real world raises new risks. If one of the apps was malicious or buggy, it could annoy or distract the user with spurious content (such as poorly placed ads), endanger the user by occluding critical information in the real world (for instance, by obscuring oncoming vehicles), or perform a denial-of-service attack on another application by occluding that application's output (for instance, a Pokémon creature that prevents the user from seeing navigation directions). A recent concept video sketches out a possible future in which AR technologies fail to address these types of threats, as shown in Figure 3. While we describe these risks in terms of an HMD platform here, we stress that they extend across platforms and domains, such as AR-enabled windshields, which—like HMDs—are fully immersive.

Thus, the high-level challenge we address in this work is how an AR platform should constrain the output behaviors of potentially buggy, malicious, or compromised applications, and how it should handle conflicts between output from multiple applications. We argue that emerging and future AR platforms must address these questions if they wish to support rich, untrusted applications that can be run simultaneously and safely used while the user interacts with the physical world (for instance, while walking or driving, not only while sitting at a desk). We observe that undesirable output is not a new concern in and of itself: recall the early days of the web, when web applications frequently opened popups and used blink tags. Browser vendors eventually constrained these undesirable behaviors by enabling popup blocking by default[13] and by obsoleting the blink tag. Unlike misbehaving applications on the early web,

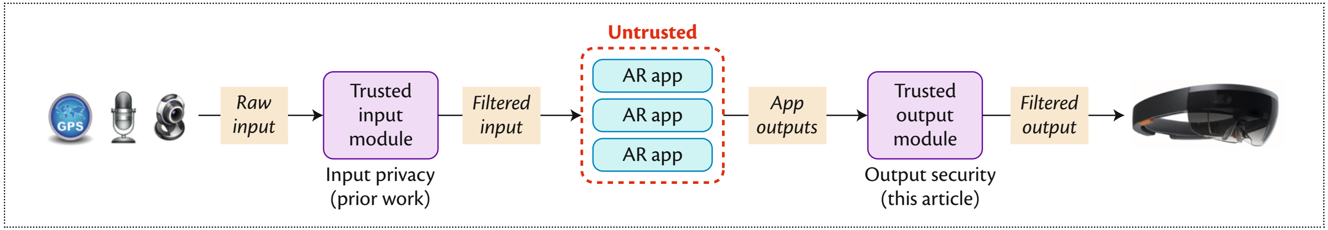

Figure 2. AR platform pipeline. AR platforms sense the real world (input), provide this sensor data to applications running on the platform, and process application requests to display virtual content (output). Prior work introduced a trusted input module to limit application access to sensitive sensor input. Our work introduces a trusted output module that constrains application output.

Figure 3. AR concept image. This concept image of an AR user on a bus could represent a possible future in which AR output remains unregulated, leaving users unable to control the intrusiveness of AR applications. Full video is available at www.theverge.com/2016/ 5/20/11719244 /hyper-reality-augmented-short-film.

the effects of problematic AR output can range from minor annoyance to direct physical harm.

The above risks inform our threat model and security goals. Specifically, we consider one or more malicious, buggy, or compromised applications that create AR content, which may intentionally or accidentally:

- obscure another application's virtual content in order to hide or modify its meaning;
- obscure important real-world content, such as traffic signs, cars, or people; or
- disrupt users physiologically, such as by startling them (for instance by suddenly creating or quickly repositioning virtual objects).

This set of threats is comparable to that used to motivate our prior work on AR output security,[12] though how to build a system to achieve these goals was then unknown.

To combat these threats, we designed Arya, an AR platform with a centralized, trusted output policy module that enforces policies on AR content. These policies aim to mitigate the above classes of threats, for instance, by preventing applications from blocking important real-world information, such as people, with AR content. Arya handles policies that can constrain when and where applications display content; it does not support policies that constrain what content is displayed (for example, a 3D animal versus a 3D rock).

We assume that Arya's operating system, drivers, and platform hardware are trusted. However, applications are not trusted by the system. Specifically, we assume that applications may be intentionally malicious, unintentionally buggy, or compromised, potentially leading to undesirable AR output. For example, an adversary might attempt to sneak an intentionally malicious application onto an open platform's app store (like the HoloLens app store), or different trusted development teams within a closed AR platform (for instance, a closed automotive AR platform) might produce applications that interact with each other unexpectedly in undesirable ways.

We also assume that Arya's operating system employs traditional, standard security best practices, such as application isolation. In this article, we focus only on threats between applications as they relate to the interaction of their AR output.

In addition, we do not address the question of how Arya-enforced AR output policies are distributed. We assume that these policies may (for example) be preloaded by the device's manufacturer, introduced by third-party sources, or set based on user preferences. We assume that policies may be buggy or malicious, and we do not require Arya to trust the sources of these policies. Thus, our design must consider the possibility of malicious or buggy policies.

Finally, we focus specifically on visual AR content, and we consider issues related to nonvisual output (for instance, haptic and audio) to be out of scope. However, the lessons in this work may apply to other output modalities as well.

Design: The Arya Platform

In our conference paper,[11] we presented Arya, an AR platform with output security as a first-class goal. Given space limitations, we focus here on the overall Arya system design. We refer readers to our conference paper for additional details and a more thorough analysis, including in-depth discussions of the key challenges that we encountered and the design tradeoffs we faced in overcoming them.

Arya System Overview

AR applications fundamentally require the ability to continuously capture and process sensor inputs, and to superimpose virtual output on the user's view of the world. For example, consider the collision warning application in Figure 1. This application must process sensor inputs to track where other cars are relative to the user, and it must dynamically generate and update visual content as appropriate, for instance, to display a warning when a collision is imminent.

Arya thus consists of the following core modules, shown in Figure 4, that it employs to both support and constrain application behaviors in the face of a dynamically changing environment:

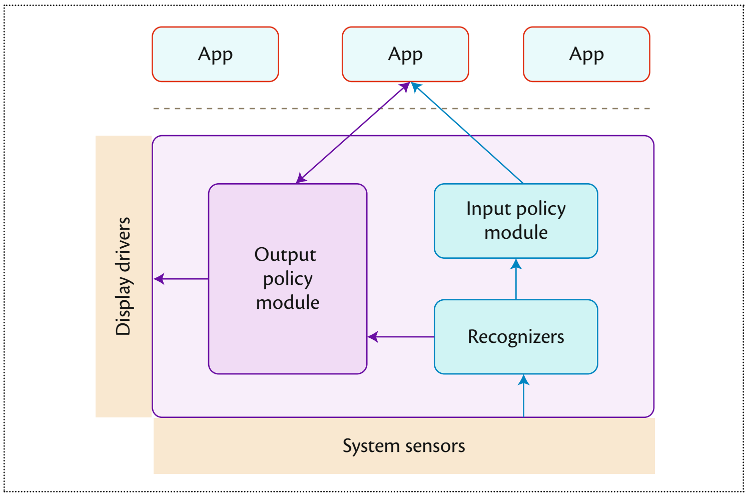

Figure 4. Overview of Arya's architecture. We designed Arya—an AR platform that consists of system sensors, recognizers, and an input policy module that filters input from the real world (based on prior work), as well as an output policy module that constrains application output (our primary contribution).

- system sensors and recognizers, to gather and interpret sensor data from the real world;
- the input policy module, to filter and dispatch this data to applications that require access;
- the output policy module, to process any new application requests to create or modify virtual content, and, if applicable, modify this virtual content based on the types of policies we introduce in this article; and
- display drivers, to display updated virtual state.

These modules are used to support applications, which may call APIs to query information about the real world and create or modify virtual objects. Arya steps through a core workflow to process application requests and produce every output video frame displayed to the user.

How Arya Handles Input

Consider again the collision warning application from Figure 1. This application must be able to detect nearby vehicles, identify where those vehicles are in relation to the user's view, and determine if a collision is imminent. One way a system might support this capability is to expose the full camera sensor feed to the application, allowing it to perform vehicle detection. However, as prior works note,[6,9,10,14] applications that can access raw, unfiltered input from the real world raise serious privacy concerns. In addition, if multiple applications need to locate vehicles in the video feed, it would be inefficient for each to implement vehicle detection separately.

To address these privacy and performance issues, prior work[6] proposed recognizers for AR platforms: OS modules that process raw sensor streams, detect specific types of information within those streams (for instance vehicles, people, faces, or planar surfaces), and expose these higher-level objects to applications. Recognizers enable a least-privilege model in which applications can be given access to only those recognized objects that they need. For example, a Pokémon game may not need a full video feed, but rather only information about planar surfaces in the user's view, to sensibly place Pokémon on horizontal surfaces.

In our work, we found that recognizers provide an additional benefit beyond their original purpose of enabling input privacy. Recognizers give Arya—and thereby Arya's output policy module—information about the user's real-world surroundings. For example, to support a policy that prevents applications from occluding people, Arya must know whether and where there are people in the user's view. Recognizers provide this information and allow Arya to enforce output policies that depend on the real world.

How Arya Handles Output

Our key research innovations for Arya center on methods to enable the OS to control the visual output of AR applications. At a high level, we do so by incorporating into the OS an output policy module, which controls and modifies AR application outputs according to policies. Arya builds on and instantiates the AR object abstraction for displaying output, proposed in our earlier work.[12] Conceptually, AR objects are OS primitives that encapsulate virtual content that applications wish to overlay on a user's view of the real world. For example, a single Pokémon creature would be an AR object in Arya, and a single application may contain many

Table 1. AR output policies.*

Identifier	Description	Applies to	Source
P1	Avoid abrupt movement of AR objects.	Car, HMD	HoloLens Developer Guidelines
P2	Place AR objects at a comfortable viewing distance from the user.	Car, HMD	HoloLens Developer Guidelines
P3	Allow the user to see the real world in the background.	Car, HMD	HoloLens Developer Guidelines
P4	Avoid content that is "head-locked" (at a fixed location in the display).	HMD	HoloLens Developer Guidelines
P5	Don't display text messages or social media while driving.	Car	NHTSA† Driver Distraction Guidelines
P6	Don't obscure pedestrians or road signs.	Car	Portland Trees Visibility Guidelines
P7	Don't obscure exit signs.	HMD	Occupational Safety and Health Regulations
P8	Disable user input on transparent AR objects.	Car, HMD	Literature on clickjacking[15]
P9	Only allow advertisements to be overlaid on real-world billboards.	Car, HMD	N/A (New)
P10	Don't allow AR objects to occlude other AR objects.	Car, HMD	N/A (New)

** This table contains a set of policies that we use to drive Arya's design. We identified existing policies from various sources (P1-P8) and, if necessary, modified them to apply to the AR context. We created two additional policies (P9 and P10) motivated by our threat model.*
† NHTSA is the US Department of Transportation's National Highway Traffic Safety Administration.

such objects. An AR object has a visual representation and associated characteristics, such as size and opacity. AR applications require the ability to create and transform these objects (for instance, by moving, rotating, or resizing them), and Arya supports these common operations.

In addition, rather than requiring that applications manually update the locations of their objects as the user moves throughout the physical world, Arya allows applications to create "world-locked" objects that are attached to real-world locations or objects, and Arya automatically updates where they are rendered in the user's display. For example, if an AR application attaches a virtual object to a real-world table, Arya can maintain this mapping, not requiring that the application explicitly update how the object is displayed as the user moves. Applications can also create "head-locked" objects that appear at a fixed location in the user's display. (Note that HoloLens similarly supports world-locked and head-locked objects. The key distinction is that Arya supports these features within the OS as part of its output management, whereas HoloLens does so at the application layer.)

The AR object model differs from the "window" display abstraction traditionally provided to applications, in which applications have full control over a contiguous rectangular screen area. A key benefit of AR objects is that they allow Arya to reason about application output and enforce policies at the granularity of individual objects. For example, if one Pokémon creature obscures a real-world person, Arya can take action against that one object (for instance, to make it transparent) without affecting the rest of the Pokémon application's output.

AR Output Policies

Central to Arya's design, and its ability to protect users from dangerous or undesirable outputs, is its support for AR output policies. To help drive our design, we developed sample output policies for both HMD and automotive AR scenarios. We drew inspiration from existing sources of guidelines, including the HoloLens developer guidelines, the US Department of Transportation guidelines for in-vehicle electronic devices, the US Department of Labor occupational health and safety regulations, and guidelines regarding the visibility of street signs. Table 1 summarizes our sample policies.

One challenge we faced was translating these abstract guidelines into enforceable policies. We first observed that our sample policies tell us only what conditions should be avoided, not what to do when the conditions are met. Thus, Arya separates the conditions under which policies apply (for example, when an AR object blocks a real-world person or is drawn too close to the user) and the mechanisms used to enforce the policies (for example, remove the AR object or make it transparent). Namely, an Arya output policy consists of two distinct components:

- a conditional predicate, or a Boolean expression that determines when a policy should be applied; and
- one or more mechanisms, or actions, that the output policy module should take when the policy's conditional predicate evaluates to true.

Determining how to express policies raised additional challenges. For example, policies comprised of arbitrary code could halt the system by performing unbounded computation, or modify AR objects in undesirable ways. Rather than allow the conditional predicates and mechanisms to consist of arbitrary code, we restrict those predicates and mechanisms in numerous ways. For example, we define a finite set of building blocks that policies can use to construct the conditional predicates. These predicates can refer to attributes of objects, which are either

- visual properties of AR objects, such as size, transparency, and speed; or
- relationships between AR objects and other virtual or real-world objects.

Another challenge we faced in designing Arya's policy mechanism framework was how to handle policies that might conflict with each other. For example, consider a policy that makes virtual objects more transparent running simultaneously with a policy that makes virtual objects more opaque. If both policies are active at once, they will create a cycle in which the object appears to flicker. To avoid such conflicts, we designed Arya's policies to be explicitly composable. Our key insight to enable policy composition is that Arya's goal in constraining AR output is to protect users from undesirable or dangerous outputs—that is, AR output that modifies the user's view of the world less is safer than output that modifies the user's view of the world more. Thus, Arya supports only policy mechanisms that move AR objects toward less intrusive states; for example, mechanisms that make objects smaller, slower, or more transparent, or that remove them or deny their creation entirely. In this way, two policies triggered under the same conditions will not yield conflicting mechanisms.

We discuss additional challenges and design tradeoffs in "Securing Augmented Reality Output,"[11] such as the nuances in how we specify policy predicates and mechanisms, where Arya evaluates policies in its workflow, as well as how much feedback Arya should provide to applications when they are impacted by policies.

Implementation

Our Arya prototype consists of several parts: an AR simulator and virtual scenes to represent the real world, the Arya core implementation (including the output policy module and infrastructure to support multiple applications), stand-alone applications that run on Arya, and AR output policies that are enforced by Arya.

AR Simulator

In practice, a full-fledged AR system has many moving parts—it continuously senses and processes real-world input, which feeds into applications as well as, in our design, the output policy module itself. However, real-world input is by its nature noisy and variable. Because the focus of our work is not on improving or evaluating AR input processing, and to support controlled, repeatable experiments, we abstract away the input-handling part of Arya for our prototype. Instead, we create an AR simulator, which consists of a virtual reality (VR) back end to represent the real world. We build on the Unity game engine, using Unity virtual environments, or "scenes," to represent the real world. This technique allows us to isolate the output management portion of the system and reliably "detect" our simulated real-world objects.

Virtual Scenes Representing the Physical World

A benefit of our AR simulator is that it easily allows us to test output policies in different Unity scenes that represent various real-world scenarios. We developed three scenes to represent HMD and automotive scenarios: an "in-home" scene (based on a prebuilt scene from the Unity Asset Store (www.assetstore.unity3d.com/en/#!/content/44784), an "AR windshield" scene, and an "office" scene. Figure 5 shows these scenes; the bare scenes, without AR applications running, are shown in the left column.

Arya Core

Up to this point, we have described only our prototyping infrastructure for representing a model of the real world. We now turn to Arya itself. We build Arya's core also on top of Unity, written in 3767 lines of C# code.

The Arya core includes recognizers, which we implement in our prototype by labeling specific "real-world"

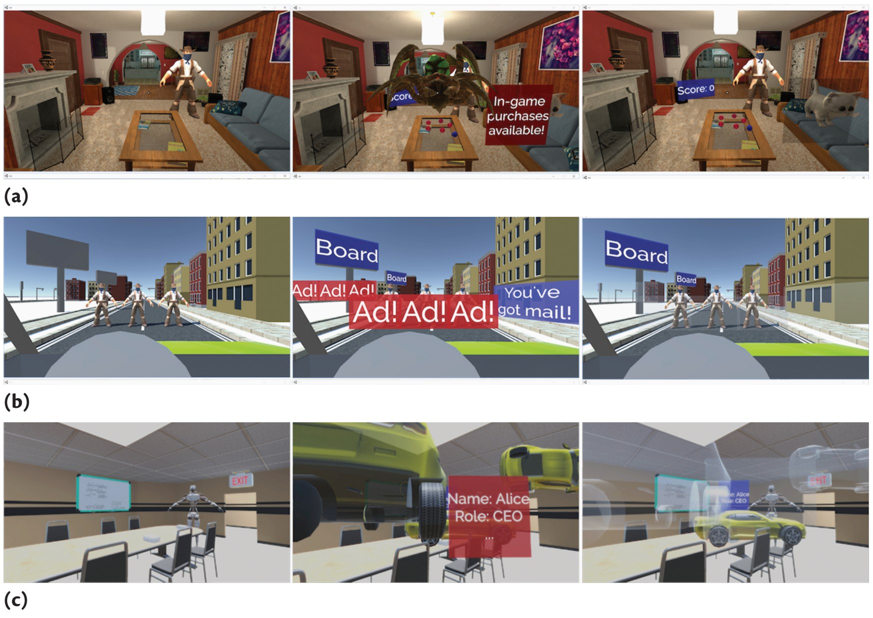

Figure 5. Case studies. These screenshots show our case study scenarios: (a) HMD in the home, (b) car windshield, and (c) HMD in the office. The left column shows the bare scenes in our Unity-based AR simulator, representing the real world without any apps running. From our prototype's perspective, everything in the bare scenes is part of the real world. The center column shows our case study apps running, exhibiting both desirable and undesirable AR output behaviors. The right column shows the result of policy enforcement, leaving only desirable AR output. Note that Unity's object opacity (or alpha value) adjustment mechanism leaves visual artifacts to outline where violating AR objects would be.

objects in our virtual scenes as objects of interest, such as people, billboards, and signs. The Arya core also includes infrastructure for running multiple AR applications on top of it, including handling multiple application threads and managing communication over local sockets. Arya exposes APIs to those applications for querying the real-world scene as well as for creating and modifying AR objects (such as `Object.Move()` and `CreateObject()`).

Applications

Multiple stand-alone applications can run atop the Arya core, to simultaneously create and interact with AR objects and augment the same "real-world" scene. Applications are isolated by running as separate OS processes, and they only interact with each other indirectly by displaying output on the same scene. Arya applications are written in C# and extend our base class `ARApplication` (889 lines of C# code).

Prototype Policies

Finally, we prototype an AR output policy framework. Policies are written as stand-alone C# modules that extend our `ARPolicy` base class and are programmatically instantiated by the Arya core. The Arya core provides a fixed set of AR object attributes (used in a policy's conditional predicate) and enforcement mechanisms that policies can employ. Table 2 details the specific case study policies we implemented. Additional attributes could be defined, as could additional mechanisms.

Evaluation

We now evaluate the efficacy of Arya's output policy module through case study applications that run within our three virtual scenes: a home, a driving scene, and an office. We designed our case study applications to exhibit both acceptable/desirable behaviors, as well as behaviors that violate one or more of our prototype policies detailed in Table 2. Figure 5 shows screenshots of our applications running in these scenes both without (center column) and with (right column) policy enforcement active. The left column shows the bare scenes, with no applications running. In the conference version of our paper,[11] we also evaluate the performance overhead introduced by Arya's prototype output policy module.

Case-Study Applications

We developed two applications per scene that test our various policies. These applications are inspired by real applications that might (or already do) exist for these emerging platforms.

Table 2. Implemented policies.*

Identifier	Conditions	Mechanisms
P1	If an AR object's speed exceeds X†	Set the object's speed to X
P2	If an AR object is within X feet of the user	Set the object's alpha value‡ to 0
P3	If an AR object occupies more than X percent of the display	Set the object's alpha value to 0
P4	If an application attempts to create a head-locked object	Deny the creation request
P5	If a user's vehicle is in motion	Set the alpha value of all applicable AR objects to 0
P6	If an AR object is occluding pedestrians or road signs	Set the object's alpha value to 0
P7	If an AR object is occluding exit signs	Set the object's alpha value to 0
P8	If an AR object's alpha value is less than X	Disable user interactions with the object
P9	If an AR object is not bounded by a real-world billboard	Set the object's alpha value to 0
P10	If an AR object is occluding another application's AR object	Set the object's alpha value to 0

** This table details the conditions under which our prototype policies are violated and the mechanisms Arya uses to enforce them. This list matches the policies in Table 1.*
† X represents a parameterized value specified by individual policies. We note that policies may be selectively applied to specific applications or groups of applications—for example, P9 may apply only to an advertising app.
‡ An object's alpha value defines how opaque or transparent it is—an object with alpha value 0 is fully transparent.

For the home scene (Figure 5a), we created a "Virtual Pet" app, which displays a world-locked virtual cat that can move independently in the user's environment. However, the application moves the cat at a distractingly fast speed through the user's view, and it displays a head-locked spider that the user cannot look away from. In addition, we built a tabletop game in which the user earns points by hitting coins with a ball. However, in-game purchase notifications block the output of other applications and may annoy the user.

For the driving scene (Figure 5b), we created an advertising application that displays targeted ads over real-world blank billboards. However, the application also displays ads throughout the rest of the user's view, potentially creating a driving hazard. In addition, we implemented a "notification" application that displays dummy text messages, a calendar, and email alerts. Unfortunately, it continues to generate distracting alerts while the car is in motion.

For the office scene (Figure 5c), we imagined a group of engineers using AR to design a new automobile (inspired by an application for HoloLens: https://www.youtube.com/watch?v=yADhOKEbZ5Q). We built an application that allows users to view their car models from different angles simultaneously. In addition, we created an application that displays information to users about their colleagues, such as their names and roles in the company. While neither of these applications exhibits intentionally malicious behavior, their outputs sometimes obscure the user's view by taking up too much of the screen, appearing too close to the user's face, or blocking out important information in the real world such as exit signs.

Security Discussion

As Figure 5 illustrates, Arya successfully allows multiple case study applications to concurrently display content while simultaneously enforcing our prototype policies to prevent malicious or undesirable output behaviors. We refer to policies by their identifiers in Table 2.

- In the home scene, P4 prevents the head-locked spider from being created. In addition, P10 prevents the in-app purchase dialog from occluding the cat (a virtual object from another application), and P1 prevents the cat from moving too fast.
- In the driving scene, P6 prevents virtual ads from obscuring real-world pedestrians, and P9 constrains

them to appearing only over real-world billboards. P5 prevents notifications from popping up while the car is in motion.

- In the office scene, P7 prevents the modeling application from blocking real-world exit signs. Meanwhile, P2 and P3 make objects that get too close to the user or take up too much space partially transparent.

Through these case studies, we confirm the ability of our policy framework to support policies that constrain a range of behaviors in different contexts. Our case studies also highlight, for completeness, an output safety risk that our current policies cannot mitigate: risks with unsafe or frightening content, such as spiders. Our policies—just like conventional web browsers, desktops, and mobile devices—do not prevent applications from displaying specific undesirable objects. This issue presents a potential avenue for future work.

Discussion

Designing a full-fledged operating system for AR platforms that supports strong security, privacy, and safety properties while enabling rich application functionality is challenging. Prior work addresses many input privacy challenges for AR, and in this work, we make significant strides toward securely handling visual output. However, many challenges and open questions remain. For example:

- *Handling noisy input sensing.* While our prototype used simulated AR environments to enable controlled output-related experiments, real AR systems will need to handle potentially noisy sensor inputs, which may confound output policy management (for instance, if a recognizer fails to detect a person).
- *More complex policies.* By supporting policy mechanisms that compose by design, Arya avoids challenges raised by potentially conflicting or flip-flopping policies. However, this design choice excludes some policy mechanisms, particularly those that move AR objects (because they might move objects to locations where they violate other policies). Future work should consider whether it is possible to design a more complex policy framework that supports policies that may conflict, for example, by employing constraint-solving approaches.
- *Nonvisual AR output.* Arya focuses on managing visual output, but as AR systems continue to evolve, we will likely see increased richness in nonvisual output, such as auditory or haptic. Thus, future work should explore how the design choices and lessons presented in this article can be applied to other types of AR output.
- *Toward a production-ready system.* Our Arya prototype yielded promising results, but additional considerations must be made to translate the core Arya ideas to production-ready systems, including addressing the above-mentioned challenges. For example, the performance overhead introduced by runtime output policy enforcement must be minimal to allow AR systems to respond to dynamic real-world changes in real time. Rather than providing a final, production-ready solution, Arya takes the first concrete steps toward identifying and mitigating AR output security risks. The insights and challenges we identify can inform future work that seeks to incorporate output management into real AR platforms.

We discuss these and other open questions further in the conference version of this article.[11]

Modifying the user's view of the world is a key feature of emerging AR applications, and left unconstrained, this ability can raise serious security and privacy risks. Now is the time to consider and address these risks. The design challenges we raise here and the solutions we propose through Arya represent a promising step toward securely augmenting reality. ■

Acknowledgments

We thank Niel Lebeck, Ada Lerner, Amy Li, Peter Ney, Lucy Simko, Anna Kornfeld Simpson, and Alex Takakuwa for valuable discussions and feedback on previous drafts; we thank Seth Kohno for his help with our AR simulator. We also thank our anonymous reviewers for their constructive feedback. This work was supported in part by the National Science Foundation under awards CNS-1513584 and CNS-1651230, and by the Short-Dooley Professorship.

References

1. I.E. Sutherland, "A Head-Mounted Three-Dimensional Display," *Proc. Fall Joint Computer Conf. American Federation of Information Processing Societies* (AFIPS), 1968, pp. 757–764.
2. M. May, "Augmented Reality in the Car Industry," LinkedIn, 1 Aug. 2015; www.linkedin.com/pulse/augmented-reality-car-industry-melanie-may.
3. K. Mizokami, "The F-35's $400,000 Generation 'Magic' Helmet Is Here," Popular Mechanics, 4 Mar. 2016; www.popularmechanics.com/military/weapons/news/a19764/the-f-35s-third-generation-magic-helmet-is-here.
4. F. Roesner, T. Kohno, and D. Molnar, "Security and Privacy for Augmented Reality Systems," *Comm. ACM*, vol. 57, no. 4, 2014, pp. 88–96.
5. T. Denning, Z. Dehlawi, and T. Kohno, "In Situ with Bystanders of Augmented Reality Glasses: Perspectives on Recording and Privacy-Mediating Technologies," *Proc.*

SIGCHI Conf. Human Factors in Computing Systems, 2014, pp. 2377–2386.
6. S. Jana et al., "Enabling Fine-Grained Permissions for Augmented Reality Applications with Recognizers," *Proc. USENIX Conf. Security*, 2013, pp. 415–430.
7. S. Jana, A. Narayanan, and V. Shmatikov, "A Scanner Darkly: Protecting User Privacy from Perceptual Applications," *IEEE Symp. Security and Privacy*, 2013; doi:10.1109/SP.2013.31.
8. J. Vilk et al., "SurroundWeb: Mitigating Privacy Concerns in a 3D Web Browser," *IEEE Symp. Security and Privacy*, 2015, pp. 431–446.
9. N. Raval et al., "What You Mark Is What Apps See," *Proc. Int'l Conf. Mobile Systems, Applications, and Services*, 2016, pp. 249–261.
10. F. Roesner et al., "World-Driven Access Control for Continuous Sensing," *ACM Conf. Computer & Communications Security*, 2014, pp. 1169–1181.
11. K. Lebeck et al., "Securing Augmented Reality Output," *IEEE Symp. Security and Privacy*, 2017, doi:10.1109/SP.2017.13.
12. K. Lebeck, T. Kohno, and F. Roesner, "How to Safely Augment Reality: Challenges and Directions," *Proc. 17th Int'l Workshop Mobile Computing Systems and Applications*, 2016, pp. 45–50.
13. R. Naraine, "Windows XP SP2 Turns 'on' Pop-Up Blocking," 18 Mar. 2004; www.internetnews.com/dev-news/article.php/3327991.
14. R. Templeman et al., "PlaceAvoider: Steering First-Person Cameras Away from Sensitive Spaces," *Network and Distributed System Security Symp.*, 2014.
15. L.-S. Huang et al., "Clickjacking: Attacks and Defenses," *Proc. 21st USENIX Security Symp.*, 2012, p. 22.

Kiron Lebeck is a PhD candidate in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. His research focuses broadly on computer security and privacy for emerging technologies, with a particular interest in augmented reality systems. Contact him at kklebeck@cs.washington.edu.

Kimberly Ruth is an undergraduate student in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. Her research currently focuses on computer security and privacy for augmented reality systems. Contact her at kcr32@cs.washington.edu.

Tadayoshi Kohno is a professor in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. He received a PhD in computer science from the University of California, San Diego. His research focuses on computer security, broadly defined. Contact him at yoshi@cs.washington.edu.

Franziska Roesner is an assistant professor in the Paul G. Allen School of Computer Science & Engineering at the University of Washington. She received a PhD in computer science and engineering from the University of Washington. Her research focuses broadly on computer security and privacy, with a particular interest in understanding and improving security and privacy for end users of existing and emerging technologies. Contact her at franzi@cs.washington.edu.