Securing Augmented Reality Output

Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, Franziska Roesner
Paul G. Allen School of Computer Science & Engineering
University of Washington
{kklebeck, kcr32, yoshi, franzi}@cs.washington.edu
https://ar-sec.cs.washington.edu

***Abstract*—Augmented reality (AR) technologies, such as Microsoft's HoloLens head-mounted display and AR-enabled car windshields, are rapidly emerging. AR applications provide users with immersive virtual experiences by capturing input from a user's surroundings and overlaying virtual output on the user's perception of the real world. These applications enable users to interact with and perceive virtual content in fundamentally new ways. However, the immersive nature of AR applications raises serious security and privacy concerns. Prior work has focused primarily on *input* privacy risks stemming from applications with unrestricted access to sensor data. However, the risks associated with malicious or buggy AR *output* remain largely unexplored. For example, an AR windshield application could intentionally or accidentally obscure oncoming vehicles or safety-critical output of other AR applications. In this work, we address the fundamental challenge of securing AR output in the face of malicious or buggy applications. We design, prototype, and evaluate Arya, an AR platform that controls application output according to policies specified in a constrained yet expressive policy framework. In doing so, we identify and overcome numerous challenges in securing AR output.**

I. Introduction

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* (e.g., 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 commercial AR efforts are relatively young, they are beginning to capture the attentions of users worldwide. For example, the wildly popular mobile AR app "Pokémon Go" [35] brought in over $600 million in revenue in its first three months after release, making it the most successful mobile game in history [44]. However, the potential of AR lies far beyond simple smartphone games, and we are beginning to see rapid growth in new AR technologies. For example, Microsoft's HoloLens HMD is now available to developers [20], Meta's second-generation HMD is available for pre-order [27], and Google has invested over $500 million in the HMD startup Magic Leap [28]. Additionally, many groups within the automotive industry are developing AR-enabled windshields to aid drivers [15, 26, 46]. Overall, interest in employing AR technologies across diverse industry sectors is increasing, with AR as a whole projected to grow into a $100 billion industry by the year 2020 [1].

Challenge: AR Output Security. Though AR technologies have the potential to deliver tremendous benefits, they also raise new privacy and security risks. A growing body of literature focuses on mitigating privacy risks that stem from applications' needs to gather *input* from the numerous sensors on AR devices, such as cameras [8, 18, 37, 39, 40, 45]. In this work, we focus instead on a complementary issue: the security risks of AR *output*, or the risks that arise from AR applications' abilities to modify a user's view of the world. Addressing these 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.

To illustrate potential security risks related to AR output, imagine driving a car with an AR-enabled windshield. The intended benefits of this technology include the ability to visibly highlight lane markers to prevent accidental lane drift, to display turn-by-turn driving directions visually overlaid on the road, and to visibly warn the driver of impending collisions — examples already showcased by industry, e.g., [17] (see also Figure 1). These tasks might run as multiple components of a single application, or as multiple, distinct applications. Without appropriate safeguards, however, the benefits of these applications can be overshadowed by risks. A malicious or buggy AR application could potentially obscure real-world pedestrians, overlay misleading information on real-world road signs, or occlude the virtual content of other AR applications, such as collision warnings or other important safety alerts. Similar issues could arise with HMDs for a user on foot. Consider, for example, an HMD application that accidentally or intentionally blocks the user's view of a tripping hazard or an oncoming car. The ability of AR content to obscure real-world objects is not hypothetical, as Figure 2 shows.

To our knowledge, no existing industry or research AR platforms are designed to mitigate the above types of output security risks. Today, it is the responsibility of the applications themselves to safely generate output and to adhere to guidelines such as those suggested for HoloLens developers [29]. For instance, these guidelines suggest that applications should not create AR content that covers too much of the user's view of the world, but HoloLens itself does not enforce this. Placing this responsibility on application developers, who may generate buggy, vulnerable, or malicious code, is problematic.

Fig. 1: ***Example AR Scenario.*** *This screenshot from Hyundai's CES demo [17] shows an AR warning overlaid on a car and the car's current speed. In future AR platforms, we expect that multiple applications will simultaneously produce output.*

Fig. 2: ***Real-World Occlusion.*** *This photo was taken by a smartphone camera through a HoloLens display (resulting in some reflective camera artifacts). It shows that virtual content displayed by HoloLens (here, a cat) can visually obscure real-world objects (also a cat).*

Furthermore, the fact that today's AR platforms cannot exert any control over the output of individual applications means they also cannot handle conflicts between the output of multiple applications. Indeed, HoloLens sidesteps this problem by not supporting multiple full-screen immersive AR applications running at once.

Our Work: Designing for Secure AR Output. We seek to change the above situation. Specifically, we design, implement, and evaluate a prototype AR platform with output security as an explicit, first-class goal. We refer to our design as Arya. 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 *policies*, such as ensuring that windshield applications cannot obscure real-world road signs or pedestrians while the car is moving. This work both identifies and overcomes numerous challenges towards designing AR systems to mitigate output security risks.

Our core design builds upon the designs of prior AR systems and includes *sensors*, such as cameras and microphones; *recognizers* [18] to detect objects, such as cars and people, from the sensed input; and an *input policy module* [40] to determine which of the sensed objects should be passed to applications, possibly with modification. The central difference in Arya is the inclusion of an *output policy module* that sits between applications and the AR system's output drivers, and that enforces policy-based constraints on application outputs. While the potential utility of an output policy module was suggested in a position paper [21], we are the first to concretely explore the feasibility of an output policy module in practice. We find that designing an output policy module is fundamentally challenging, and requires identifying and answering key design questions, such as how to express desired output policies, how to enforce those policies, and how to handle policy conflicts.

We identify and overcome these challenges through the iterative design, implementation, and evaluation of Arya and our Arya prototype. To drive our design, we develop a set of case study output policies based on existing policies drawn from several sources, including the HoloLens developer guidelines [29] and guidelines for the visibility of road signs [6]. For example, we use a guideline that real-world trees should not block road signs to inspire a policy that virtual objects should not block real-world road signs. To support such policies, we design an *AR output policy specification framework* that allows policy writers to specify both (1) a condition under which the policy is violated (e.g., when a virtual object blocks a real-world person) and (2) an action to take (e.g., make the offending virtual object partially transparent). We carefully constrain this policy framework to support composable policies and to limit the potential performance or other impacts of buggy or malicious policies. We do not specify where policies come from in this work — they may come from the device manufacturer itself or other sources.

We develop our prototype atop the Unity game engine [47], an environment for creating interactive 3D content. To evaluate the output management portion of Arya through controlled experiments that simulate different real-world contexts, we develop virtual Unity scenes rather than using real-world sensor input. Our scenes represent HMD and car windshield AR scenarios, and we develop a set of case study applications that run within these scenarios. We demonstrate that our prototype can support the policies we identify and prevent corresponding undesirable situations in our case studies. We conduct a performance evaluation consisting of both microbenchmarks and a full system evaluation, and we find that the performance impact of policy enforcement in even our unoptimized prototype is acceptable. Our prototype played a central role in iteratively driving the design of Arya, and our design choices and evaluation findings provide lessons for future AR system designers.

Contributions. In summary, we contribute the following:

1) *AR Output Security*: We address, for the first time, the fundamental challenge of securing AR output in the face of malicious or buggy applications. We design

Arya, an AR platform that can control visual application output according to output policies. Our design prevents single applications from modifying the user's view of the world in undesirable ways, and it securely supports multiple applications simultaneously augmenting the user's view of the world — unlike existing AR platforms such as HoloLens, which do neither.

2) *AR Output Policies*: We develop a policy specification framework for defining output policies that is designed to provide desirable properties (e.g., to limit performance impact and support composable policies). Through our design process, we uncover and overcome fundamental challenges in realizing the above vision, including how to specify and enforce policies, and how to handle conflicting policies. Despite its restrictions, we demonstrate that our framework can support real policies from multiple sources, such as the HoloLens developer guidelines and U.S. Department of Transportation guidelines for in-vehicle electronic devices.
3) *Prototype, Evaluation, Lessons*: We prototype Arya on top of the Unity game engine and develop case study applications and policies for both HMD and automotive AR scenarios. We conduct benchmark and full system evaluations, finding the performance of policy enforcement acceptable. From our experiences, we surface lessons and recommendations for future AR systems.

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. It is critical that we identify and address AR security challenges while the technologies are still young and designs are not yet set in stone. Our work lays a technical foundation to support future AR security efforts, and to enable immersive single- and multi-application AR platforms whose potential benefits are not overshadowed by risks due to buggy, malicious, or compromised AR application output.

II. Context

We begin by providing additional context on the capabilities of AR and its rising commercial presence. Emerging commercial AR platforms support fundamentally new types of applications that can respond contextually to input from a user's ever-changing environment, and that can directly alter the user's perception of his or her world with visual, auditory, or haptic output. Since today's AR devices primarily rely on immersive visual feedback, we focus most of our concrete discussions on visual output, though we note that similar issues may apply to other output modalities (e.g., audio or haptic).

Many industries are beginning to leverage emerging AR technologies for diverse purposes. For example, Microsoft's HoloLens is being used by NASA's Jet Propulsion Laboratory to guide astronauts through complex tasks [2], and by the Israeli military to manipulate terrain models and monitor troop positions [3]. Along with Microsoft, companies such as Meta [27] and Magic Leap [25] are also developing sophisticated AR headsets. Additionally, the U.S. military has shown increasing interest in AR [23, 32, 41], and researchers and companies within the automotive industry are exploring AR-enabled windshields and dedicated HMDs to aid drivers. Haeuslschmid et al. [15] describe a broad taxonomy of AR windshield applications grounded in existing literature, ranging from safety-oriented apps (e.g., highlighting lane markers to warn a driver of accidental lane drift) to navigation apps (e.g., path finding with 3D navigation arrows). Recent demos from Hyundai [17] (shown in Figure 1) and Continental [7] demonstrate the capabilities of early-stage AR windshields, and organizations such as BMW [4] and Honda Research [46] continue to push the boundaries of automotive AR.

Though emerging AR platforms and applications hold great promise, these technologies are still young and under active development. For example, HoloLens has only released its developer edition to date, and it is limited by the fact that only one immersive AR application can run at a time. However, we expect that future AR users will wish to allow multiple applications to simultaneously augment their view of the physical world, e.g., one application that translates and superimposes text in real time, one that shows calendar and email alerts, and one that runs a game (such as Pokémon Go). Similarly, in future versions of the automotive example in Figure 1, we envision that the collision warning capability and the speedometer capability might be different applications written by different development teams within the automotive company, or even by third-party app providers.

III. Motivation and Threat Model

In addition to their novel opportunities, AR applications have a unique ability to impact users' perceptions of the real world in undesirable or harmful ways. To understand these risks, consider the popular mobile AR app Pokémon Go. While this game is a relatively simple smartphone app today, it provides a taste of how emerging platforms like HoloLens will be able to capture the attention of users [42]. In contrast to smartphones, HMDs provide continuous, fully immersive experiences by enveloping a user's entire field of view. With these emerging HMD platforms, we envision that a user may wish to multitask while playing a game like Pokémon Go — for example, by using an app that overlays walking directions to nearby restaurants, or by using a labelling app to recognize and point out nearby social media contacts. To reap the full benefits of these apps, the user will need to use them while actively moving about and interacting with the real world.

The interaction of multiple AR apps with each other and with the user's view of the real world raises risks. If one of the apps were malicious or buggy, it could (a) annoy or distract the user with spurious content (e.g., poorly-placed ads), (b) endanger the user by occluding critical information in the real world (e.g., by obscuring oncoming vehicles), or (c) perform a denial of service attack on another application by occluding that application's output (e.g., a Pokémon creature that prevents the user from seeing navigation directions). Indeed, 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

Fig. 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 available at http://www.theverge.com/2016/5/20/11719244/hyper-reality-augmented-short-film*

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 (e.g., 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 [33] and by obsoleting the blink tag. Unlike misbehaving applications on the early web, the effects of problematic AR output can range from minor annoyance to direct physical harm.

Threat Model. 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.
- *Disrupt the user physiologically*, such as by startling them (e.g., by suddenly creating or quickly repositioning virtual objects).

This set of threats is comparable to that used to motivate prior work on AR output security [21], though how to build a system to achieve these goals was then unknown.

To combat these threats, we design 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, e.g., 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 (e.g., 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 (e.g., 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, e.g., application isolation. In this work, we focus only on threats between applications as they relate to the interaction of their AR output.

Additionally, we do not address the question of how the AR output policies that Arya enforces are distributed. We assume that these policies may (for example) be pre-loaded 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 non-visual output (e.g., haptic, audio) to be out of scope. However, the lessons we surface through this work may apply to other output modalities as well.

IV. Design: Arya

We now present the design of Arya, an AR platform architecture with output security as a first-class goal. In designing Arya, we identify and address new, fundamental design challenges that future AR platforms must consider if they wish to constrain AR application output. We begin with a high-level overview of Arya in Section IV-A, summarized in Figure 4, before describing its constituent components and the technical challenges they address in greater depth.

A. 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. Consider the collision warning application in Figure 1. This application must know when the user moves too close to another car so that it can display a warning whenever the user is at risk for a collision. However, the user's view of the real world is constantly in flux — the user may change lanes, or other cars may move in front of the user. Furthermore, applications may need to dynamically generate and update visual content in response to these changes — e.g., to display a warning when a collision is imminent. When this content is generated, Arya may also need to modify it to ensure that the warning does not

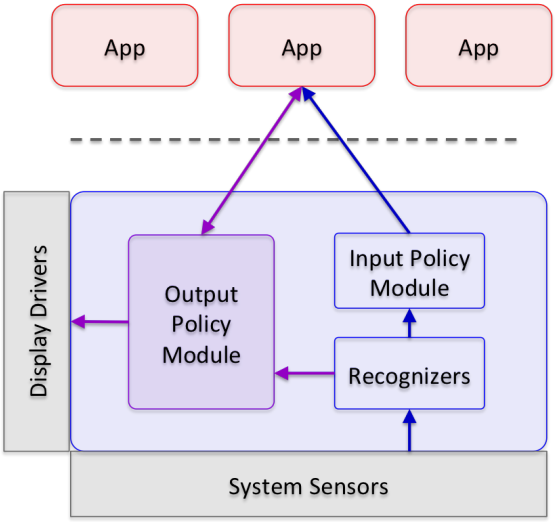

Fig. 4: ***Overview of Arya's Architecture.*** *We design Arya, an AR platform that consists of (1) system sensors, recognizers, and an input policy module that filters input from the real world, based on prior work (e.g., [18, 36, 40, 45]) and (2) an output policy module that constrains application output. The design of the output policy module is the primary contribution of this work.*

occlude any pedestrians that stumble into the road, or impede the driver's view of the car that he or she is about to hit.

Arya 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:

- *System Sensors and Recognizers*, to gather and interpret sensor data from the real world.
- The *Input Policy Module*, to filter and dispatch these 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 paper.
- *Display Drivers*, to display updated virtual state.

These modules are used to support applications running on Arya that 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. We first discuss how Arya incorporates prior work to handle input in Section IV-B, before turning to our primary contribution — output management — in Section IV-C.

B. 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 (e.g., [18, 37, 40, 45]), applications that can access raw, unfiltered input from the real world raise serious privacy concerns. Additionally, 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 [18] proposed *recognizers* for AR platforms: OS modules that process raw sensor streams, detect specific types of information within those streams (e.g., 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 this work, we find that recognizers provide an additional benefit beyond their original purpose of enabling input privacy. Recognizers give Arya itself — 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.

C. Output

Recall that our goal in designing Arya is to allow 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. Before describing these policies and their enforcement in detail in upcoming sections, we describe here the visual output abstractions that Arya exposes to applications.

Foundation: Displaying and Constraining Visual Output. Arya builds on and instantiates the *AR object abstraction* for displaying output, proposed in prior work [21]. 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 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 (e.g., by moving, rotating, or resizing them), and Arya supports these common operations.

Additionally, 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.[1]

Note that 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 (e.g., to make it transparent) without affecting the rest of the Pokémon application's output.

We now turn to the remainder of our design. We present our key design questions, describe the challenges involved in creating an output policy module that constrains AR application output, and surface key design decisions made along the way.

1) Specifying AR Output Policies

Output policies broadly serve to protect AR users from deceptive, discomforting, or harmful content. While AR technologies are still quite young, concretely exploring the policy design space grounded in today's technologies allows us to begin to identify key challenges for future AR systems and to surface initial solutions. Thus, given an output policy module that constrains virtual content in the form of AR objects, our first design question is the following:

Design Question: *How can we translate abstract guidelines into concrete policies that the output policy module can enforce in practice?* To help drive our design around this question, we developed sample output policies for both HMD and automotive AR scenarios. In addition to creating our own policies, we draw on existing sources of guidelines for the relevant scenarios, including the HoloLens developer guidelines (which are suggestions, not technically enforced constraints), the U.S. Department of Transportation guidelines for in-vehicle electronic devices, and guidelines regarding the visibility of street signs. These policies are summarized in Table I.

The first observation we make based on our case study policies in Table I is that they tell us only what conditions should be *avoided*, not *what to do* when the conditions are met. For example, we would like Arya's output policy module to prevent applications from creating objects that are too close to the user, take up too much of the user's field of view, block pedestrians, etc. However, existing guidelines do not specify what actions the output policy module should take if an application violates one of these policies. For example, possible actions to enforce policies may include removing, moving, or modifying (e.g., making more transparent) an app's AR objects. We consider these options further below.

Design Decision: Separate Policy Conditions and Mechanisms. The above observation raises an opportunity: the conditions under which policies apply (e.g., 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 (e.g., remove the AR object or make it transparent) can be specified independently and composed as desired.

[1]HoloLens similarly supports world-locked and head-locked objects [30]. The key distinction is that Arya supports these features within the OS as part of its output management, while HoloLens does so at the application layer.

Specifically, we define AR output policies to consist of two distinct components:

1) A *conditional predicate*, or a boolean expression that determines when a policy should be applied.
2) One or more *mechanisms*, or actions that the output policy module should take when the policy's conditional predicate evaluates to true.

The next design question we face is then the following:

Design Question: *How should policy conditions and mechanisms be expressed?* The most flexible approach would be to allow conditions and mechanisms to consist of arbitrary code, which would clearly support a wide range of policies. However, arbitrary policy code raises several concerns. The first is performance: in the worst case, an arbitrarily-defined policy could halt the system by performing unbounded computation. The second is unexpected results due to buggy or untrusted policies: if policy mechanisms can arbitrarily modify applications' AR objects, then buggy policies could pose the same risks as buggy apps themselves in the worst case.

Design Decision: Restrict Policies. Due to the challenges raised by arbitrary policies, we instead develop an *explicitly restricted policy framework* that requires policies to combine options from a well-defined set of parameterized conditions and mechanisms supported by Arya. Though this construction is limited by design, we find that it is flexible enough to express the set of desirable policies we developed ourselves and drew from other sources (see Table I).

Policy Conditions. We develop a finite set of building blocks that policies can use to construct conditional predicates. Specifically, we allow policies to refer to *attributes* of objects. We define attributes to be either (1) visual properties of AR objects, such as size, transparency, and speed, or (2) relationships between AR objects and other virtual or real-world objects. For example, relational attributes include `DistanceFromUser()` or `IsOccluding(type)`, where "type" refers to a class of objects against which to check for occlusion (virtual objects or specific types of real-world objects detected by Arya's recognizers, such as people). For non-boolean attributes, a policy's condition is then formed by comparing one or more attributes of an AR object to parameter values specified by the policy — for example, "`if DistanceFromUser() < 10 meters`".

Finally, we allow policy conditions to depend not only on the attributes of AR objects, but also on global contextual information. For example, a policy may depend on properties of the user's platform (e.g., if a user's car is in motion) or other contextual information (e.g., time of day).

Policy Mechanisms. Policy mechanisms are more challenging to design, because they involve not just deriving boolean results, but modifying application behaviors. As mentioned above, possible mechanisms that Arya might support include deleting applications' AR objects (or not allowing them to be created in the first place), modifying them (e.g., to change their transparency or size), or moving them (e.g., away from blocking another object). In experimenting with different possible mechanisms, we identified the following challenge:

Identifier	Description	Applies To	Source
P1	Avoid abrupt movement of AR objects.	Car, HMD	HoloLens Developer Guidelines [29]
P2	Place AR objects at a comfortable viewing distance from the user.	Car, HMD	HoloLens Developer Guidelines [29]
P3	Allow the user to see the real world in the background.	Car, HMD	HoloLens Developer Guidelines [29]
P4	Avoid content that is "head-locked" (at a fixed location in the display).	HMD	HoloLens Developer Guidelines [29]
P5	Don't display text messages or social media while driving.	Car	NHTSA Driver Distraction Guidelines [50]
P6	Don't obscure pedestrians or road signs.	Car	Portland Trees Visibility Guidelines [6]
P7	Don't obscure exit signs.	HMD	Occupational Safety and Health Regulations [49]
P8	Disable user input on transparent AR objects.	Car, HMD	Literature on clickjacking (e.g., [16])
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)

*TABLE I: **AR Output Policies.** 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. Note that NHTSA (the source of P5) is the U.S. Department of Transportation's National Highway Traffic Safety Administration.*

Challenge: Conflicting Policies. Since multiple policies may be triggered at once, certain combinations of policy mechanisms may conflict with each other or create a cycle. For example, consider one policy that moves an object away from blocking a person, but causes it to block a road sign, thereby triggering another policy. Or consider a policy that reduces an object's transparency at the same time as another policy attempts to increase its transparency.

We can address this challenge in one of two ways. First, we could design a method to handle policy conflicts when they arise. However, this raises many additional challenges — for example, what should be done if the conflict cannot be resolved, whether conflict resolution can be performed quickly enough, and how non-conflicting but cyclic policies should be handled. Though there may well be solutions to these challenges (as we elaborate in Section VII), in this work we take another approach: we design policy mechanisms such that they *cannot conflict* in the first place.

Design Decision: Composable Policy Mechanisms. It is not immediately obvious how to design policy mechanisms that are composable yet sufficiently flexible to express meaningful policies. However, we observe the following: the goal of our AR output policies is ultimately to ensure that AR applications cannot modify the user's view of the world in dangerous or undesirable ways. Thus, policies should constrain application output to be *less intrusive*, so that the result is closer to an unmodified view of the real world. Based on this observation, we choose to support only policy mechanisms that move AR objects towards a less intrusive state — for example, mechanisms that make objects smaller, slower, or more transparent, or that remove them or deny their creation entirely.

Designing policy mechanisms in this way gives us our desired property of composability. For example, consider a case in which one policy wishes to set an object's opacity to 50%, and another to 30% (more transparent). As stated, we cannot satisfy both policies at once — the object cannot have both 50% and 30% opacity. However, if we return to the notion that the goal of a policy is to modify attributes to be less intrusive — in this case, more transparent — we can consider these policies as specifying thresholds. That is, the first policy wishes to enforce a *maximum* of 50% opacity, and the second a *maximum* of 30%. Formulated this way, these policies compose: setting the object's opacity to 30% satisfies both policies. Thus, given some set of thresholds set by different policies, Arya takes the most restrictive intersection (i.e., the attribute values that result in the least intrusive state) and enforces these thresholds on AR objects.

In addition to supporting composable policies, this design also ensures that we can no longer encounter a situation in which policies flip-flop, with one making an object more transparent and the other making the object less. In the above example, the subsequent activation of a third policy specifying a higher maximum opacity (e.g., 60%) would not change the most restrictive active threshold (30%).

This design decision intentionally disallows mechanisms that might result in cyclic policy violations or lead to complex constraint solving, but that may sometimes be desirable (e.g., automatically repositioning AR objects). We discuss possible approaches that future work must explore to support such policies in Section VII.

Finally, we note that malicious or buggy policies *can* still result in applications being able to display less content, thus impacting application functionality. However, due to the composable properties of our polices, they cannot, by definition, result in *more* intrusive output. That is, Arya is fail-safe in the face of malicious or buggy policies.

2) Enforcing AR Output Policies

Now that we have determined how policies are specified, we turn our attention to how they are enforced by Arya's output policy module. The algorithms in Figure 5 detail policy condition checking and mechanism enforcement at different points within Arya, as we will introduce below.

Although we have thus far discussed policies as though they always apply to all applications and objects, we note that they can be enforced more granularly. For example, policies can be enforced selectively on the objects of specific applications or categories of applications (e.g., entertainment or safety-oriented apps). However, we do not focus on this granularity for the below discussion, instead assuming a more general situation in which policies do apply.

Design Question: *At what points in its workflow should Arya evaluate policies?* The first natural place to check and enforce policies is when applications attempt to create, move, or

Algorithm 1 Example policy checked on API

```
1: procedure CREATE(AR OBJECT a, AR OBJECT SET A)
2:     for each On-Create Policy p do
3:         deny ← p.Evaluate(a)
4:         if deny then DenyCreation(a) ; return
5:     Create a ; A ← A ∪ a
```

Algorithm 2 Per-frame policy enforcement

```
1: procedure UPDATE
2:     Update mapping of real world
3:     for each AR Object a ∈ A do
4:         for each Per-Frame Policy p do
5:             p.Evaluate(a)
6:         PolicyModule.EnforceThresholds(a)
7:     M ← Incoming API requests
8:     for each m in M do
9:         ProcessRequest(m)
10:    E ← Pending callback events
11:    for each e in E do
12:        SendEvent(e, targetApp)
13:    finally: Render AR Objects
```

Algorithm 3 Example attribute-modifying API call

```
1: procedure SETALPHA(AR OBJECT a, VALUE alpha)
2:     thresh ← a.AlphaThreshold
3:     if thresh < alpha then alpha ← thresh
4:     a.alphaValue = alpha
```

Fig. 5: ***Policy Enforcement.*** *These algorithms give pseudocode for how Arya checks and enforces policies (1) on API calls and (2) during the per-frame update loop. The thresholds set when a policy is enforced are respected (3) when object attributes are modified. Policy enforcement is detailed in Section IV-C2.*

modify their AR objects. For example, consider a policy with a condition such as "`if obj.size > X`" and a mechanism such as "`obj.SetAlpha(0.2)`" (i.e., a policy that makes large objects semi-transparent). This policy's condition can be checked, and its mechanism enforced, when the application calls `CreateObject()` or `ResizeObject()`. Similarly, a policy that prevents head-locked objects (in a fixed position of the user's display) can be evaluated and enforced on the call to `CreateObject()`. Algorithm 1 presents example pseudocode for policy evaluation on the `CreateObject()` API call; Arya handles other APIs similarly.

Challenge: Handling Relational Policies. Through our implementation experience with different policies, we find that only checking and enforcing policies on API calls is insufficient when those policies depend on relationships between objects, which may be virtual objects or detected real-world objects. Consider the example of a policy with the condition "if an AR object is occluding a real-world person" and the mechanism "set its opacity to 0.2" — or, in pseudocode, "`if obj.isOccluding(person) then obj.setAlpha(0.2)`". Clearly, this condition could be triggered when an application attempts to create or move its AR objects in a way that obscures a real-world person. However, even without explicit action by an application, changes in the real world (such as a person walking in front of the user) could result in a policy violation.

Now consider a related policy that refers only to virtual objects: "if an AR object is occluding another AR object, set its opacity to 0.2". At first glance, it seems that this policy *can* be enforced on API calls, i.e., when an application creates or moves virtual objects. However, suppose the user changes his or her viewing angle or moves around the physical world. In this case, Arya automatically updates the rendered locations of world-locked virtual objects without explicit API calls from the applications. As a result, objects that were previously not occluding each other may now be violating the policy.

Thus, as these two examples show, Arya needs to be able to enforce policies that depend on relationships between objects *independently of actions taken by applications*. This observation leads to the following design decision:

Design Decision: Check Relational Policy Conditions at Regular Intervals. To account for changes in the real world that may affect policy decisions, such as the user's position and viewing angle, Arya cannot wait for applications to explicitly change their objects. Instead, it must continuously monitor policy conditions that relate to real-world objects (e.g., on a per-frame basis[2]). Thus, on every frame, Arya gathers information from its input recognizers (e.g., to determine if and where there are people in the current view of the real world) and notes the current state of all AR objects. This information is then used to evaluate policies such as the examples above. Once all per-frame policy conditions have been evaluated on an object, Arya enforces the respective policy mechanisms by finding the most restrictive intersection of attribute thresholds and applying them. In the above examples, Arya would set the opacity of the violating object to 0.2. Algorithm 2 details Arya's per-frame policy enforcement workflow. However, we must now consider the following:

Design Question: *How do relational policies that influence specific attributes (e.g., opacity) interact with API calls that modify the same attributes?* For example, consider again the policy which reduces the opacity of AR objects that occlude real-world people to 0.2. What happens if, after this policy is enforced, the application calls `SetAlpha(1.0)` to make that object opaque? If Arya naively evaluates the policy on the current frame before processing the API call, the object will — at least temporarily, for one frame — violate the policy. Such a temporary violation, particularly if the application calls `SetAlpha(1.0)` repeatedly, could nevertheless be disruptive to the user. On the other hand, if Arya processes the API call before enforcing the per-frame policy, it creates additional overhead by needing to roll back the results of the API call.

Design Decision: Decouple Threshold Setting and Enforcing. To avoid both of the above problems, we decouple setting a threshold value for an attribute from enforcing that

[2]Our design considers per-frame checking for relational policies, but it generalizes to other regular intervals. For example, Card et al. [5] suggest that a 100ms interval may be sufficient.

threshold. In the above example, the policy sets an opacity threshold of 0.2 when it is evaluated per-frame. That threshold is immediately enforced, i.e., the object's opacity is reduced. However, to avoid temporary violations, those thresholds are *also* enforced on any API calls processed in the same frame. That is, when Arya handles the `SetAlpha(1.0)` API call, it respects the current opacity threshold for that object, not exceeding 0.2. This process is detailed in Algorithm 3, which shows an example for the `SetAlpha()` API; other attribute-modifying API calls are handled similarly.

3) When Policy Violations Cease

Having considered how policies are specified and how they are enforced, we turn to a final question:

Design Question: *What should Arya do when a previously-enforced policy is no longer violated?* That is, when an AR object that was modified due to a policy ceases to violate said policy, how should those modifications be reverted?

An initially appealing approach is to have Arya itself manage the reversal of policy enforcement. For example, if Arya reduced an AR object's opacity to 0.2 in response to a policy, Arya should also return that object's opacity back to normal when the policy condition is no longer violated (e.g., when the object no longer occludes a real-world person). A benefit of this approach is the loose coupling between AR objects and policies, allowing applications to operate oblivious of any active policies. However, this design raises the following challenge:

Challenge: Policy Impact on Application State. When considering an object attribute, what constitutes a "normal" value is unclear — is it the value of that attribute at the time the policy was first violated? That state may no longer be valid when the policy violation ceases. Is it the application's current expected value of that attribute, supposing it has continued to update what it would be without any policy effects? That may work in many cases, but in other cases, the application may have made different decisions if it had known about the policy violation. For example, an application whose objects are made transparent due to a policy may wish to remove the objects in question. These considerations illuminate a key tradeoff between application flexibility and more straightforward, policy-oblivious behavior.

Design Decision: Inform Applications About Policies. We choose to inform applications when their objects start or stop violating policies, so they can react appropriately. Under this model, if an app whose object is modified by a policy wishes to, for example, remove that object or display an error message to the user, it can do so. Similarly, this design allows applications flexibility in determining appropriate attribute values after an object stops violating a policy, rather than having Arya revert object attributes oblivious to application semantics.

In choosing to deliver information to apps about when their objects violate policies, we uncover an additional challenge:

Challenge: Privacy Risks. Sharing too much information about policy violations with applications can compromise privacy. Recall that, for privacy reasons (and building on prior work [18]), an application may not have access to a full video feed but rather limited recognizer inputs, e.g., planar surfaces. Now suppose, for example, that when an application's object is made transparent because it overlapped a real-world pedestrian, Arya triggered a callback to the application informing it not only how its AR object was affected but also which policy was violated. While sharing the details of the violated policy could be useful (e.g., allowing the application to move it object to stop violating the policy), it also raises privacy concerns. Specifically, it can reveal information to applications about real-world objects (e.g., that a pedestrian is present) or about other applications' AR objects.

Design Decision: Provide Limited Feedback to Applications. To mitigate this privacy risk, Arya does not share the full details of policy violations with applications. Instead, it informs applications only when attribute thresholds on its objects change (e.g., when an object is made transparent, or when the maximum allowable alpha value increases when a policy is no longer violated), so that it can react appropriately. However, Arya does not provide any details about the policy condition that triggered the threshold change.

4) Design Summary

In summary, we identified key design questions regarding how to specify AR object policies and avoid conflicts between policies (Section IV-C1), how to enforce policies (Section IV-C2), and what to do when objects cease to violate policies (Section IV-C3). To address these questions and the challenges they raise, we developed an output policy specification framework in which policies consist of restricted, composable conditions and enforcement mechanisms, with privacy-conscious feedback to applications when violations occur or cease.

We consider the design questions and challenges that we uncovered through this process to be contributions in and of themselves. While our proposed solutions meet our security goals, future AR system designers may wish to make different design choices. Our work surfaces a number of challenges and tradeoffs that must be considered, which we hope will help guide potential alternate design paths.

V. Implementation

We now describe our prototype implementation of Arya. Developing our prototype gives us the opportunity to deeply explore and evaluate Arya's AR output policy module, and iteratively feeds back into our design process. Our 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), standalone applications that run on Arya, and AR output policies that are enforced by Arya. We detail these components in turn.

AR Simulator. In practice, a full-fledged AR system has many moving parts — crucially, 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, as we discuss

in Section VII. Even if we had perfect sensor hardware and sensor data processing algorithms, we would still like to evaluate in controlled, possibly hard-to-stage scenarios (e.g., while driving).

Since the focus of our work is not on improving or evaluating AR input processing (a topic of other research efforts, e.g., [9, 22, 31]), and to support controlled 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) backend to represent the real world. This approach is similar to driving simulators commonly used in other research, e.g., [48].

Specifically, rather than outfitting our prototype with real hardware sensors, 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. AR applications running on Arya can create virtual objects to place into these scenes, and Arya's output policy module can regulate those objects given information about the fully-specified underlying VR world.

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. Specifically, we developed three scenes to represent HMD and automotive scenarios: an "in-home" scene,[3] an "AR-windshield" scene, and an "office" scene. These scenes are shown in Figure 6; the bare scenes, without AR applications running, are shown in the left column of that figure. We emphasize that these scenes represent the real world, and that no virtual content created by AR applications is shown in the bare scenes.

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[4]. Loading this core into a new scene requires only a few user interface actions within the Unity editor. While Arya interfaces with our virtual scenes, it is largely modularized.

The Arya core 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()`).

We implement recognizers in our prototype by labeling specific "real-world" objects in our virtual scenes as objects of interest, e.g., people, billboards, and signs. This information about the real world, as well as the state Arya keeps about applications' AR objects created and modified through its APIs, feeds into Arya's output policy module. This module enforces policies on application output, as detailed in Section IV-C2.

[3]We augmented a pre-built scene, "Brian's House", purchased from the Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/44784

[4]We use the CLOC tool for calculating lines of code: https://github.com/AlDanial/cloc/releases/tag/v1.70

Application Interface. Our prototype supports multiple standalone applications running atop the Arya core, which can simultaneously create and interact with AR objects and augment the same "real-world" scene. Applications are isolated by running as separate OS processes, such that their only interaction is implicitly by augmenting the same "reality."

Arya applications are written in C# and extend our base class `ARApplication`. This base class contains 889 lines of C# code and provides the infrastructure for communicating with the Arya core over local sockets to make API calls (e.g., to create or modify objects). We describe case study applications that we implemented for our evaluation in Section VI.

Prototype Policies. Finally, we prototype an AR output policy framework. Policies are written as standalone C# modules that extend our `ARPolicy` base class and are programmatically instantiated by the Arya core. As described in Section IV, policies follow a well-defined structure consisting of a condition and a mechanism. The Arya core provides a fixed set of AR object attributes (used in conditions) and enforcement mechanisms that policies can employ. Table II details the specific case study policies we implemented. We stress that the conditions and mechanisms we chose to implement are not the only possible options that Arya can support. Additional attributes could be defined, as could additional mechanisms that meet our composability criteria (moving objects towards "less intrusive" states). For example, our most complex attribute (determining if one AR object occludes another object) consists of only 49 lines of code, suggesting that developing new attributes could be easily done.

VI. Evaluation

Our evaluation goals are two-fold. First, we seek to evaluate Arya's ability to support and enforce a variety of policies from different sources. Second, since policy enforcement is on the critical path for rendering output, we measure the performance overhead introduced by our prototype's output policy module. Our results suggest that Arya is a promising approach for constraining AR output — not only does it successfully address, for the first time, many output security issues, but it also does so with reasonable performance. We use these results to surface additional lessons and recommendations for future AR platform developers.

A. Case Studies: Policy Expressiveness and Effectiveness

We evaluate the efficacy of Arya's output policy module through case study applications that run within our three virtual scenes, described in Section V: a home, a driving scene, and an office. We design our case study applications to exhibit both (a) acceptable or desirable behaviors, as well as (b) behaviors that violate one or more of our prototype policies detailed in Table II. Figure 6 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.

Case-Study Applications. We developed two applications per scene that test our various policies. Our focus is to exercise

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

TABLE II: ***Implemented Policies.*** *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 I. 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 only apply to an advertising app.*

our output policies, and thus we did not implement complex application-level logic. Nevertheless, these applications are inspired by real applications that might (or already do) exist for these emerging platforms.

HMD in the Home. For the home scene (top row of Figure 6), 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. Additionally, we built a tabletop game[5] in which the user increases their score by hitting coins with a ball. However, the application pops up in-game purchase notifications that block the output of other applications and may annoy the user.

AR Windshields. For the driving scene (center row of Figure 6), 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. Additionally, we implemented a "notification" application that displays dummy text message, calendar, and email alerts. Unfortunately, it continues to generate distracting alerts while the car is in motion.

HMD in the Workplace. For the office scene (bottom row of Figure 6), we imagine a group of engineers using AR to design a new automobile.[6] We built an application that allows users to view their car models from different angles simultaneously. Additionally, we created an application that displays information to users about their colleagues, such as their names and roles in the company. While both of these applications do not exhibit 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 illustrated in Figure 6, Arya successfully allows multiple case study applications to concurrently display content while simultaneously enforcing our prototype policies to prevent malicious or undesirable output behaviors. Specifically, referring to policies by their identifiers in Table II:

- In the home scene, P4 prevents the head-locked spider from being created. Additionally, 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.

These case studies exercise all but one of the policies we implemented (Table II). The exception is P8, which disables user input on obscured AR objects. Though we implemented this policy, we cannot exercise it, because our prototype is designed to focus on generating output and hence lacks meaningful user input for application interactions.

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.

B. Performance Evaluation

Arya's output policy module directly mediates content that applications wish to display and thus lies on the critical path for rendering. As such, the output policy module should incur minimal overhead. While our prototype implementation is not optimized or representative of a full-fledged AR system, analyzing its performance can nevertheless shed light on possible output bottlenecks and other considerations that must go into implementing an output policy module in a production system.

Our case-study applications successfully exercise our prototype policies, but they contain relatively few AR objects. To identify potential bottlenecks, we next analyze the performance

[5]Inspired by https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial.

[6]Inspired by an application for HoloLens: https://www.youtube.com/watch?v=yADhOKEbZ5Q.

Fig. 6: ***Case Studies.*** *These screenshots show our case study scenarios: HMD in the home (top), car windshield (center), and HMD in the office (bottom). 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 scene 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 alpha adjustment mechanism leaves transparency artifacts to outline where violating AR objects would be.*

of the output policy module under heavier workloads, i.e., when there are many objects present. We first profile the performance of our output policy module in the absence of our application communication infrastructure to isolate the performance impact of our policies. We then analyze our communication infrastructure and conduct a full-system evaluation.

1) Profiling the Output Policy Module

We begin by profiling our prototype's output policy module without the overhead of application communication. To isolate the impact of the output policy module, we create a simple evaluation scene containing several objects (a "person", a "billboard", and an "exit sign"). Rather than having a separate application process create and update AR objects, we instead programmatically trigger API calls directly in Arya's core on a per-frame basis. From the output policy module's perspective, these requests appear to come from an actual application. This setup simulates application behaviors but eliminates any performance impact of the communication infrastructure and allows us to focus on the output policy module itself. This methodology also allows us to ensure the same amount of work occurs each frame, enabling repeatable experiments.

Our primary performance metric for profiling the output policy module is the frame rate, or average frames-per-second (FPS), of Arya's Unity backend. Since Arya's core functions (handling API calls and enforcing policies) operate on a per-frame basis, extra overhead introduced by the output policy module directly decreases the frame rate, making FPS a meaningful metric. For each data point in our measurements, we calculated the average FPS over a 30 second interval (after an initial 10 second warm-up period), repeating each trial 5 times. We conduct two evaluations with this experimental setup: first, we compare the individual performance of the policies we implemented, and then we investigate policy performance as we scale the number of virtual objects in the scene.

Individual Policy Performance. We begin by trying to understand the performance impact of our individual policies relative to a baseline scene without any policy enforcement. These results are shown in Tables III and IV.

In designing this experiment, our goal is to fully tax the system, such that differences between policies become visible. To do so, we simulate the following application behaviors: we create N overlapping objects directly in front of the user, and move each object a small amount every frame. For these experiments, we chose N objects such that the baseline would be under load — i.e., less than 60 FPS, which is considered a standard for smooth gameplay in many PC video games [13] — so that we could see the effects of policies. We experimentally determined that $N = 500$ objects would give us a baseline

	Baseline	P1	P2	P3	P6	P7	P8	P10
Avg FPS	51.4	51.3	48.0	39.2	49.0	43.7	43.8	32.3
Std Dev	1.2	1.3	1.1	1.5	0.4	1.6	1.1	1.8

*TABLE III: **Profiling Policy Performance.** As described in Section VI-B1, we calculate the average frame rate of the Arya core with different active policies, compared to a baseline with no active policies. Policy identifiers in this table match those in Tables I and II. In our experimental scenes, we load the system by having 500 objects that each move once per frame, and each tested policy is violated on every frame. Results are averaged over five 30-second trials.*

	Baseline	P4
Avg FPS	4.6	57.7
StdDev	1.0	2.0

	Baseline	P9
Avg FPS	32.6	30.7
StdDev	1.0	1.2

*TABLE IV: **Profiling Policy Performance.** For two policies, we use a different experimental setup, with different baseline measurements, than used in Table III. For P4, which acts on the `CreateObject()` API, we create and delete objects every frame rather than moving them. For P9, we create virtual objects locked to a real-world billboard. Since the object-locking functionality itself incurs overhead (independently of policies), we generate a separate baseline. As in Table III, results are averaged over five 30-second trials.*

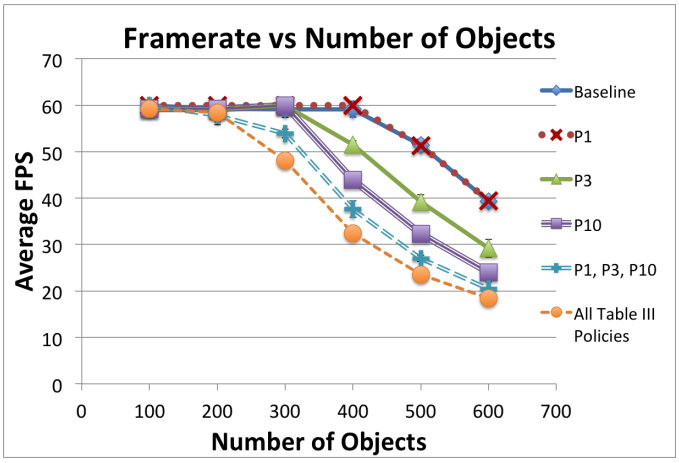

*Fig. 7: **Performance with Multiple Policies and Scaling AR Objects.** We investigate the performance impact of combining multiple policies and how that impact scales with increasing numbers of AR objects in the scene. We find that the performance overhead of multiple policies is less than the sum of the overhead from those policies individually, and that the performance hit of adding AR objects (unrelated to policies) dominates the impact of policy enforcement.*

frame rate of less than 60 FPS.

We designed the scene such that every frame, each virtual object violates each policy we implemented (see Table II), though we only activate and evaluate one policy at a time. Two of our policies required slightly different experimental setups to trigger violations: P4 requires that the baseline setup repeatedly attempt to create objects each frame, and P9 requires the baseline setup to contain objects that are locked to real-world objects (in this case, billboards). The results for these two policies are in Table IV, and the caption further details the specific experimental setups.

Tables III and IV show the results of these experiments. We observe a range of performance impacts across our different policies. For example, P1 (which limits the speed at which objects can move) and P2 (which makes objects too close to the user transparent) incur virtually no additional overhead over the baseline. On the other hand, P10 (which makes virtual objects that obscure other virtual objects transparent) incurs an almost 20 FPS hit.

A key observation is that *the complexity of object attributes directly influences policy performance*. For example, P1 simply sets a threshold on objects' movement speeds, which is easily checked and enforced when an application calls `object.Move()` with a speed parameter. On the other hand, P10 incurs more overhead because it must detect virtual objects that occlude others in every frame, requiring costly raycasting operations. This lesson suggests that optimizing attribute computations and intelligently caching information will be critical for such a scheme to work in practice.

This lesson is further supported by our experience applying preliminary optimizations to P10. Initially, P10 incurred significant overhead due to redundant raycasting operations between overlapping objects, resulting in an average frame rate under 2 FPS. However, by optimizing P10 to not repeat computation on AR objects that the policy has already acted upon, we significantly improved its performance. This suggests that pursuing policy optimizations can have a great impact.

Finally, we note that P4, a policy that denies certain `OnCreate()` calls, actually *improved* performance over the baseline. This is a result of the baseline scene repeatedly creating and deleting headlocked AR objects, in contrast to P4 simply denying the requests. Thus, we observe that policies that deny object creation could also be used as a denial-of-service protection against applications attempting to create many objects.

Policy Performance Scaling with AR Objects. The above benchmark provides a single snapshot of how our policies compare, with a fixed number of virtual objects (500). However, we also wish to understand (1) how policy performance scales as the number of active AR objects that violate them increases, and (2) how performance is affected by multiple simultaneously running policies.

Using the same experimental setup from Table III, we compare the baseline scene to several policies, as well as combinations of policies, as we vary the number of active AR objects present. We select the policies for this experiment based on the results in Table III, choosing our best performing policy (P1) and two worst-performing policies (P3 and P10). Figure 7 shows the results of this experiment. Note that we cap the maximum FPS at 60 using Unity's `Application.targetFrameRate` feature.

Our results reveal several interesting lessons. First, *policy overhead is not additive*. The performance hit incurred by several policies combined, even those that leverage different attributes, is less than the sum of their overheads individually. This finding is promising, since in practice, multiple policies may indeed be active at once. Even if the list of policies

	1 App	2 Apps	3 Apps	4 Apps
Avg Msgs/App/Second	1808	1020	646	508
Std Dev	221	115	251	141

*TABLE V: **Arya Message Throughput.** To inform our choice of parameters for a full system evaluation (shown in Figure 8), we first characterize the performance of our unoptimized application communication infrastructure, by which applications use local sockets to communicate with the Arya core to make API calls. The results are averaged over five 30-second trials.*

increases, we expect overlapping work between policies. For example, the cost of loading objects in memory could be amortized across multiple policies, and multiple policies may require similar computations about objects.

Second, we observe that *the performance impact of additional virtual objects dominates the impact of policies*. That is, as the number of AR objects increases, the frame rate of the baseline with no policies drops below 60 FPS, scaling with the number of objects. Although the frame rate with multiple active policies drops below 60 FPS more quickly, the impact of multiple policies scales with number of AR objects similarly to the baseline, after the initial performance hit of activating any policies. This is perhaps not surprising: more complex applications will run more slowly. However, the fact that the performance impact of policy enforcement does not become increasingly worse with more AR objects is promising.

2) Full System Evaluation

Our above experiments isolate the performance impact of the output policy module and evaluate it with respect to varying numbers of AR objects and policies. However, we also wish to understand the impact of the output policy module in the face of multiple prototype applications simultaneously running on Arya. Since our primary focus was on the output policy module, other elements of the system — specifically, its handling of multiple application threads and local socket communications — are unoptimized. To isolate the performance impacts of these unoptimized components, we first conduct a microbenchmark evaluation to profile Arya's application communication infrastructure. Using the results of this microbenchmark, we choose parameters for a meaningful full system evaluation such that we do not hit bottlenecks due to communication and accidentally mask the impact of the output policy module.

Communication Microbenchmark. We first measure the throughput of Arya's message processing infrastructure. We connect application processes to Arya over local sockets, after which the applications saturate the connections with messages, which Arya then processes as fast as it can. Table V summarizes the message throughput of Arya with increasing numbers of concurrently running applications, where one message corresponds to one API call. As we increase the number of applications, the number of messages Arya can process per application decreases. This result is expected, since each application runs as a separate process, and communication between Arya and each app run on separate threads.

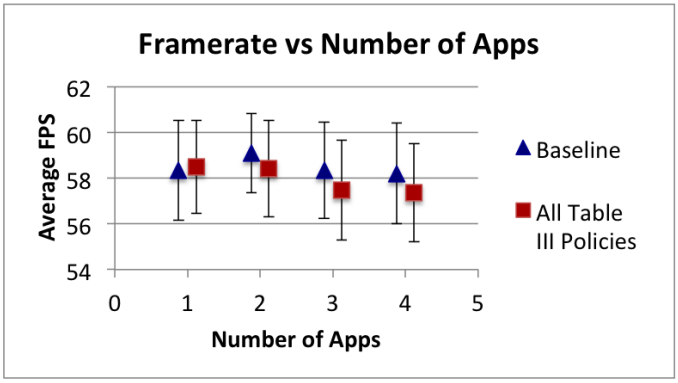

*Fig. 8: **Full System Evaluation.** This graph shows the results, in terms of Arya's core frame rate, of running 1-4 applications with 7 active policies, compared to a baseline with no active policies. As described in Section VI-B2, the total number of objects is fixed at 48, split evenly across the number of applications in a given trial. Note that this graph's y-axis does not start at 0, so that the small differences in performance are visible. We find that under this reasonable workload, the performance impact of policy enforcement is minimal.*

Putting It All Together. Finally, we evaluate our full prototype. We compare the average FPS under workloads with different numbers of applications communicating over sockets, and with many active policies. As before, we designed a scene in which there are multiple virtual objects, each moving once per frame, and we calculate the average FPS over a 30 second interval.

We use the results of our socket microbenchmark to determine a realistic workload — i.e., a total number of AR objects — that will avoid communication bottlenecks. We fix the total number of AR objects for this experiment at 48, evenly split across the number of running applications (1-4). Each application calls the `object.Move()` API on each of its objects approximately 30 times per second. We arrive at 48 objects based on the results from Table V: Arya can support up to about 1800 messages per second, and $48 \times 30 < 1800$, and it is evenly divided by 1, 2, 3, and 4 (number of apps we test). While 48 objects is much less than the 500 we used in our profiling experiments above, those experiments were specifically designed to tax Arya, whereas 48 represents a more reasonable workload for applications. For example, our case study apps consisted of only a handful of objects each. Additionally, in practice, apps may not call APIs on each of their objects continuously, though we do so in our experiments.

We compared this workload, with all seven policies from Table III active and continuously violated, to the baseline. Our results are shown in Figure 8. The error bars represent the standard deviation of 5 trials. The result is promising: we find that *under this realistic, 48-object workload, the performance impact of policy enforcement is negligible over the baseline*. Whereas our earlier profiling of the output policy module highlights bottlenecks (e.g., attributes that are expensive to compute) under load, our full system evaluation suggests that even our unoptimized prototype can handle multiple applications and multiple policies under a realistic workload.

VII. 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 towards securely handling visual output. However, many challenges remain. We step back and reflect on these challenges, and we make recommendations for designing future secure AR systems.

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. Input noise may confound output policy management (e.g., if a recognizer fails to detect a person). Thus, future work must explore how to mitigate risks from noisy input — e.g., considering how to deal with ambiguity and probabilities, and how to determine appropriate defaults. For example, recognizers may need to output confidence values — e.g., confidence that there is a person in the video feed — and the output policy module may need to use confidence values across multiple frames to make determinations.

Constraint-Solving Policy Framework. 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 (since they might move objects to locations where they violate other policies). Some systems may wish to support such policies: for example, automatically repositioning a safety dialog on an AR windshield to ensure that it remains visible but does not obscure pedestrians. Future work should consider whether it is possible to design a more complex policy framework that supports policies that may conflict. One approach may be to allow applications to express AR object attributes as constraints rather than fixed values (e.g., specifying several acceptable locations where an AR objects may be displayed), giving the output policy module the responsibility of solving those constraints in the face of all active policies. However, such a system would still need to answer the question of what to do when a given set of constraints cannot be solved. Prior work has considered similar constraint-solving approaches for laying out UIs in more traditional platforms (e.g., tablets or phones) [12]. Techniques from this work may be applicable here, though the AR context also raises new challenges (e.g., the potential for constant constraint solving due to rapid changes in the real world).

Application Prioritization. With many applications potentially competing to display output that is subject to a variety of policies, we argue that Arya could benefit from a prioritization scheme that favors certain applications over others. While not the focus of this study, we observe, for example, that a safety-critical application might receive priority over a game if their outputs conflict or if the user encounters a dangerous situation.

API Extensibility. Arya hides low-level data from untrusted applications, providing high-level abstractions for applications to receive input (recognizers [18]) and to display output (AR objects). While this model effectively restricts the capabilities of malicious or buggy applications, it may also present flexibility challenges for honest applications (similar to the input flexibility challenges faced in [18]). A key question is thus how Arya should expose mechanisms for adding additional functionality without compromising the security of the system. While also not the focus of our study, we observe that an extensibility model analogous to OS device drivers, with modules developed by reputable third parties, could facilitate more flexible options for application developers.

Non-Visual AR Output. Arya focuses on managing visual output, but as AR systems continue to evolve, we will likely see increased richness in non-visual output, such as auditory or haptic. Thus, future work should explore how the design choices and lessons presented in this paper can be applied to other types of AR output. We expect that some challenges and design choices will be similar (e.g., a condition/mechanism-based policy framework) while others will differ. For example, beyond blocking certain audio output entirely, are there other, less strict mechanisms that may be viable (similar to partial transparency of visual content)?

Low-Level Support for AR Objects. Arya relies on the AR object abstraction, by which an application's visual output consists of multiple non-rectangular regions of pixels, rather than a single rectangular window. The traditional window abstraction is deeply embedded in today's operating systems and their interactions with graphics and display hardware. In our prototype, these issues were below the abstraction level of our implementation, which was built atop the Unity game engine. However, future work — and certainly non-prototype AR systems interfacing more directly with hardware — will need to consider how the AR object abstraction can and/or should be incorporated into lower-level design choices.

VIII. Related Work

The computer security research community has recently identified the need to address security and privacy for emerging AR systems (e.g., [8, 39]). Arya leverages recent work in this area, particularly work on limiting AR application access to potentially sensitive sensor data (e.g., [18, 19, 36, 40, 45]). Also related, SurroundWeb [51] allows applications to project onto surfaces in a room, but considers primarily privacy concerns (not sharing video information about the room) rather than output security. PrePose [11] supports defining new input gestures (e.g., hand motions) while mitigating privacy and security threats from sensing applications.

To date, most work on AR security and privacy has focused on input-related issues. By contrast, Arya focuses on output. It builds on prior work that identified the need to securely handle AR output and leverages the AR object model from that work [21], but dives much deeper, uncovering and addressing additional fundamental challenges.

MacIntyre et al. [24] proposed an earlier version of AR objects, but without a focus on security. Greenberg et al. [14] discuss "dark" design patterns in ubiquitous proximity-based

computing more generally; several of these patterns (e.g., "captive audience") are related to the AR output threats we consider. Ng-Thow-Hing et al. [34] discuss design guidelines for automotive AR applications, some of which (e.g., limiting distraction) can be enforced by Arya's policies.

Researchers have previously considered the challenge of constraining and securing application output in more traditional platforms (e.g., [10, 38, 43]). While AR platforms can build on lessons from these prior platforms, the existence of virtual 3D objects that can overlay on the user's view of the real world raises new challenges.

IX. Conclusion

Immersive augmented reality technologies, such as head-mounted displays like Microsoft's HoloLens or automotive windshields, are becoming a commercial reality. Though the computer security research community has begun to address input-related risks with emerging AR platforms, little has been done to address *output risks*. Our work considers these risks — for example, buggy or malicious applications that create virtual content that obscures the user's view of the real world (or the virtual content from other applications) in undesirable or unsafe ways.

To address these risks, we design, implement, and evaluate Arya, an AR platform that supports multiple applications simultaneously augmenting the user's view of the world. Arya's primary contribution is the design of an *output policy module* that constrains AR application output according to policies (e.g., preventing virtual content from obscuring a real-world person). We identify and overcome numerous challenges in designing an AR output policy specification and evaluation framework that supports composable, effective, and efficient policies. We evaluate our prototype implementation of Arya with prototype policies drawn from various sources. We find that Arya prevents undesirable behavior in case study applications, and that the performance overhead of policy enforcement is acceptable even in our unoptimized prototype.

Now is the time to consider the security and privacy risks raised by emerging AR technologies. Modifying the user's view of the world is a key feature of AR applications, and left unconstrained, this ability can raise serious risks. We argue that future AR platforms must consider and address these issues if they are to safely and securely support multiple, simultaneously running applications as well as continuous use of immersive AR as the user moves about the physical world. The design challenges we raise in this paper, and the solutions we propose through Arya, represent a promising step towards secure AR output.

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. Finally, we thank Freckles for posing for Figure 2. 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] ABI research shows augmented reality on the rise with total market worth to reach $100 billion by 2020. https://www.abiresearch.com/press/abi-research-shows-augmented-reality-rise-total-ma/.

[2] How virtual, augmented reality helps NASA explore space. http://www.siliconvalley.com/2016/04/04/how-virtual-augmented-reality-helps-nasa-explore-space/.

[3] G. Ackerman and D. Bass. Israeli army prepares augmented reality for battlefield duty. https://www.bloomberg.com/news/articles/2016-08-15/microsoft-s-hololens-technology-adopted-by-israeli-military.

[4] R. Baldwin. Mini's weird-looking AR goggles are actually useful, Apr. 2015. http://www.engadget.com/2015/04/22/bmw-mini-qualcomm-ar/.

[5] S. Card, T. MORAN, and A. Newell. The model human processor- an engineering model of human performance. *Handbook of perception and human performance.*, 2:45–1, 1986.

[6] City of Portland. Trees & Visibility, Safety, & Clearance. https://www.portlandoregon.gov/trees/article/424262.

[7] Continental. Augmented reality head-up display. https://www.youtube.com/watch?v=3uuQSSnO7IE.

[8] L. D'Antoni, A. Dunn, S. Jana, T. Kohno, B. Livshits, D. Molnar, A. Moshchuk, E. Ofek, F. Roesner, S. Saponas, et al. Operating system support for augmented reality applications. *Hot Topics in Operating Systems (HotOS)*, 2013.

[9] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4), 2012.

[10] J. Epstein, J. McHugh, and R. Pascale. Evolution of a trusted B3 window system prototype. In *IEEE Symposium on Security and Privacy*, 1992.

[11] L. S. Figueiredo, B. Livshits, D. Molnar, and M. Veanes. PrePose: Security and privacy for gesture-based programming. In *IEEE Symposium on Security and Privacy*, 2016.

[12] K. Gajos and D. S. Weld. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interface*, 2004.

[13] 60 FPS on Consoles. http://www.giantbomb.com/60-fps-on-consoles/3015-3223/.

[14] S. Greenberg, S. Boring, J. Vermeulen, and J. Dostal. Dark Patterns in Proxemic Interactions: A Critical Perspective. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, pages 523–532. ACM, 2014.

[15] R. Haeuslschmid, B. Pfleging, and F. Alt. A design space to support the development of windshield applications for the car. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5076–5091, New York, NY, USA, 2016. ACM.

[16] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson. Clickjacking: Attacks and defenses. In *21st USENIX Security Symposium*, 2012.

[17] Hyundai. Hyundai augmented reality demonstration - CES 2015. https://www.youtube.com/watch?v=iZg89ov75QQ.

[18] S. Jana, D. Molnar, A. Moshchuk, A. M. Dunn, B. Livshits, H. J. Wang, and E. Ofek. Enabling fine-grained permissions for augmented reality applications with recognizers. In *USENIX Security*, 2013.

[19] S. Jana, A. Narayanan, and V. Shmatikov. A Scanner Darkly: Protecting user privacy from perceptual applications. In *IEEE Symposium on Security and Privacy*, 2013.

[20] A. Kipman. Microsoft HoloLens welcomes six new countries to the world of mixed reality, Oct. 2016. https://blogs.windows.com/devices/?p=258014?ocid=newsletter_ema_omc_hol_october_nonowners_MarketExpansion.

[21] K. Lebeck, T. Kohno, and F. Roesner. How to safely augment reality: Challenges and directions. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2016.

[22] X. Li, F. Flohr, Y. Yang, H. Xiong, M. Braun, S. Pan, K. Li, and D. M. Gavrila. A new benchmark for vision-based cyclist detection. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[23] M. A. Livingston, L. J. Rosenblum, D. G. Brown, G. S. Schmidt, S. J. Julier, Y. Baillot, J. E. Swan II, Z. Ai, and P. Maassel. Military applications of augmented reality. In *Handbook of augmented reality*, pages 671–706. Springer, 2011.

[24] B. MacIntyre, M. Gandy, S. Dow, and J. D. Bolter. Dart: a toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th ACM symposium on User Interface Software and Technology*, pages 197–206. ACM, 2004.

[25] Magic Leap. https://www.magicleap.com/#/home.

[26] M. May. Augmented reality in the car industry, Aug. 2015. https://www.linkedin.com/pulse/augmented-reality-car-industry-melanie-may.

[27] https://www.metavision.com/.

[28] R. Metz. Magic Leap: A startup is betting more than half a billion dollars that it will dazzle you with its approach to creating 3-D imagery. MIT Technology Review, 2015. https://www.technologyreview.com/s/534971/magic-leap/.

[29] Microsoft. Designing for mixed reality. https://developer.microsoft.com/en-us/windows/holographic/designing_for_mixed_reality.

[30] Microsoft. HoloLens: Coordinate systems. https://developer.microsoft.com/en-us/windows/holographic/coordinate_systems.

[31] A. Milan, L. Leal-Taixé, I. D. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016.

[32] K. Mizokami. The F-35's $400,000 third-generation "magic" helmet is here, 2016. http://www.popularmechanics.com/military/weapons/news/a19764/the-f-35s-third-generation-magic-helmet-is-here/.

[33] R. Naraine. Windows XP SP2 turns 'on' pop-up blocking, 2004. http://www.internetnews.com/dev-news/article.php/3327991.

[34] V. Ng-Thow-Hing, K. Bark, L. Beckwith, C. Tran, R. Bhandari, and S. Sridhar. User-centered perspectives for automotive augmented reality. In *IEEE International Symposium on Mixed and Augmented Reality*, 2013.

[35] http://www.pokemongo.com/.

[36] N. Raval, A. Srivastava, K. Lebeck, L. Cox, and A. Machanavajjhala. Markit: Privacy markers for protecting visual secrets. In *Workshop on Usable Privacy & Security for wearable and domestic ubIquitous DEvices (UPSIDE)*, 2014.

[37] N. Raval, A. Srivastava, A. Razeen, K. Lebeck, A. Machanavajjhala, and L. P. Cox. What you mark is what apps see. In *MobiSys*, 2016.

[38] F. Roesner and T. Kohno. Securing embedded user interfaces: Android and beyond. In *USENIX Security Symposium*, 2013.

[39] F. Roesner, T. Kohno, and D. Molnar. Security and privacy for augmented reality systems. *Communications of the ACM*, 57(4):88–96, 2014.

[40] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. J. Wang. World-driven access control for continuous sensing. In *ACM Conf. on Computer & Communications Security*, 2014.

[41] A. Rosenblum. Augmented reality glasses are coming to the battlefield, 2015. http://www.popsci.com/experimental-ar-glasses-offer-marines-hands-free-intel.

[42] D. Rubino. Microsoft's Nadella weighs in on Pokémon Go, HoloLens, and the bright future for AR. http://www.windowscentral.com/microsofts-nadella-weighs-pokemon-go-hololens.

[43] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS trusted window system. In *13th USENIX Security Symposium*, 2004.

[44] D. Takahashi. Pokémon go is the fastest mobile game to hit $600 million in revenues. http://venturebeat.com/2016/10/20/pokemon-go-is-the-fastest-mobile-game-to-hit-600-million-in-revenues/.

[45] R. Templeman, M. Korayem, D. Crandall, and A. Kapadia. PlaceAvoider: Steering first-person cameras away from sensitive spaces. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[46] C. Tran, K. Bark, and V. Ng-Thow-Hing. A left-turn driving aid using projected oncoming vehicle paths with augmented reality. In *5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2013.

[47] https://unity3d.com/.

[48] University of Iowa. The National Advanced Driving Simulator. http://www.nads-sc.uiowa.edu/.

[49] U.S. Department of Labor, Occupational Safety and Health Administration. Occupational Health and Safety Standards: Maintenance, safeguards, and operational fea-

tures for exit routes. .

[50] U.S. Department of Transportation, National Highway Traffic Safety Administration. Visual-Manual NHTSA Driver Distraction Guidelines For In-Vehicle Electronic Devices (Docket No. NHTSA-2010-0053), 2010. http://www.distraction.gov/downloads/pdfs/visual-manual-nhtsa-driver-distraction-guidelines-for-in-vehicle-electronic-devices.pdf.

[51] J. Vilk, A. Moshchuk, D. Molnar, B. Livshits, E. Ofek, C. Rossbach, H. J. Wang, and R. Gal. SurroundWeb: Mitigating privacy concerns in a 3D web browser. In *IEEE Symposium on Security and Privacy*, 2015.