When the User Is Inside the User Interface: An Empirical Study of UI Security Properties in Augmented Reality

Kaiming Cheng, Arkaprabha Bhattacharya, Michelle Lin, Jaewook Lee, Aroosh Kumar, Jeffery F. Tian, Tadayoshi Kohno, Franziska Roesner
Paul G. Allen School of Computer Science & Engineering, University of Washington
https://ar-sec.cs.washington.edu
{kaimingc, arkabhat, mlin88, jaewook4, arkumar, jefftian, yoshi, franzi}@cs.washington.edu

Abstract

Augmented reality (AR) experiences place users inside the user interface (UI), where they can see and interact with three-dimensional virtual content. This paper explores UI security for AR platforms, for which we identify three UI security-related properties: Same Space (how does the platform handle virtual content placed at the same coordinates?), Invisibility (how does the platform handle invisible virtual content?), and Synthetic Input (how does the platform handle simulated user input?). We demonstrate the security implications of different instantiations of these properties through five proof-of-concept attacks between distrusting AR application components (i.e., a main app and an included library) — including a clickjacking attack and an object erasure attack. We then empirically investigate these UI security properties on five current AR platforms: ARCore (Google), ARKit (Apple), Hololens (Microsoft), Oculus (Meta), and WebXR (browser). We find that all platforms enable at least three of our proof-of-concept attacks to succeed. We discuss potential future defenses, including applying lessons from 2D UI security and identifying new directions for AR UI security.

1 Introduction

Extensive past research and practice have considered user interface (UI) security for two-dimensional screens (desktop, browser, and mobile), e.g., clickjacking attacks that trick the user into interacting with UI elements [33,37,44,54,67,82], information leakage via the user interface [27,42], and isolating UI components from other entities [13,74,92]. In this paper, we explore UI security in emerging augmented reality (AR) platforms.[1] AR immerses the user *inside* a three-dimensional user interface — including, in some contexts, the real world itself — in contrast to the user merely observing it from the outside.

Scope and Threat Model. UI-level security for AR includes potential attacks on: (1) the user's *perception of the physical world*, (2) the virtual world or other *virtual content*, and (3) the user's *interactions with virtual content*. Regarding the first threat model, recent work has begun to study security and privacy for emerging AR platforms more generally, including some UI-related issues related to attacks on physical world perception. For example, it has discussed or demonstrated attacks in which malicious AR content is used to obscure important real-world (or virtual) content [43, 61] and side-channel attacks that allow malicious applications to infer information about the user's physical surroundings [70, 99, 101]. In this work, we consider a threat model where multiple entities might be interacting within the AR UI, such as third-party embedded code (e.g., a library) running inside an AR application in which the embedded code (e.g., the library) seeks to compromise a property of the AR application or vice versa. As the ecosystem continues to develop, we envision our threat model extending to future multi-user/multiple AR applications simultaneously augmenting the user's view of the world.

Given our focus on multiple principals, we thus particularly consider threat models (2) and (3), i.e., from one principal on another principal's virtual content and on the user's interaction with another principal.

AR UI Security Attacks and Properties. Given this problem scope, we prototype several multi-principal UI security attacks on currently available AR platforms. These include:

1. A *clickjacking attack* on ARKit (Apple), where a malicious AR application component tricks the user into interacting with another component's virtual object. This attack is achieved by placing both objects at the same 3D coordinates and taking advantage of ARKit's inconsistency about which object is visible and which receives user input.
2. A *user input denial-of-service attack* on Hololens (Mi-

[1] We use the term "augmented reality (AR)" to refer to technologies that place virtual content in a user's view of a real-world environment, whether embedded in it or overlaid on it. Other works may use other terms to refer to the same or related concepts, including mixed reality (MR) and extended reality (XR). We focus on AR and while we believe some findings might be possible in VR too, we are not in a position to fully clarify all differences between AR and VR.

crosoft), where a malicious AR application component blocks the user from interacting with another or *any* virtual object. This attack is achieved by surrounding the target AR object in an invisible 3D box and/or entirely surrounding the user with an invisible 3D virtual box that captures all user input. This attack is possible on platforms that support invisible objects that are allowed to receive inputs.
3. An *input forgery* attack in ARCore (Google), where a malicious AR application component impersonates the user's interaction with virtual objects, even when they are not within the user's field of view. This attack is achieved by programmatically generating synthetic user interactions, taking advantage of the absence of input provenance, i.e., the ability to verify the origin of input.
4. A *object-in-the-middle attack* on Oculus (Meta), where a malicious AR application component snoops on the user's interactions with another component's virtual object. This attack is achieved by a combination of (a) an invisible object that intercepts the user's intended input, and (b) synthetic user input that is then dispatched to the original target object.
5. An *object erasure* attack in WebXR (browser), where a malicious AR application component uses an invisible virtual object to cause an underlying victim virtual object to disappear completely. This attack is possible due to WebXR's method for rendering invisible objects.

These proof-of-concept attacks are possible because of the way each platform implements three *specific UI-related properties* that we identified: What happens when multiple virtual objects are placed at the same 3D physical coordinates (*"Same Space"* property)? (2) How do "invisible" virtual objects work (*"Invisibility"* property)? (3) Do platforms allow synthetic user input (*"Synthetic User Input"* property)?

Empirical Evaluation of Current AR Platforms and SDKs. After identifying these properties and demonstrating their security and privacy implications through the proof-of-concept attacks, we conduct a *systematic empirical investigation* of how current AR platforms and SDKs handle them. Specifically, we conduct experiments with ARCore (Google), ARKit (Apple), Hololens (Microsoft), Oculus (Meta), and WebXR (browser). We find inconsistencies in how current platforms handle the AR UI security properties that we identify and observe that many current implementations of these platforms enable attacks such as those we describe above.

Towards Future Defenses. Our results highlight the necessity for emerging AR platforms to implement UI-level security precautions in their designs and implementations. However, defenses are not necessarily straightforward. In some cases, we recommend that platforms adopt known approaches from 2D UI security (e.g., adding user input provenance information [11]), though we found that no current platform has implemented this feature. In other cases, AR-specific approaches may need to be devised [63] or AR-specific tradeoffs considered (e.g., aligning the physics and rendering engines around UI security properties and handling UI isolation in a 3D, interleaved context [60]) when 2D mitigation fails to work.

Contributions. We contribute the following:

1. We identify **three AR UI properties that have security and/or privacy implications** and provide criteria for evaluating them (Section 3).
2. Based on these properties, we demonstrate **five proof-of-concept AR UI security attacks** that are possible on today's AR platforms (Section 4).
3. We present **results of an empirical analysis** of these AR UI security properties in five commercially available AR platforms and associated SDKs: ARCore, ARKit, Hololens, Oculus, and WebXR (Section 5).

Finally, we reflect on **foundations for future defenses**, including known defenses that have not to date been applied to AR platforms and SDKs, as well as potential novel defensive directions (Section 6).

Disclosure. We have reported all of our findings to Apple, Google, Meta, Microsoft, WebXR, and Unity.

2 Background and Motivation

We begin with background on AR technologies, AR UI, and prior work on AR security and privacy to motivate our work.

Augmented Reality. AR technologies — technologies that seamlessly blend the physical and the digital worlds — are receiving increasing attention from both academia and industry. Once considered niche yet expensive research prototypes, AR devices are becoming more available and affordable, and mainstream smartphones and browsers are now supporting AR functionalities. Technologies like Microsoft's Hololens 2 [16], Meta's Oculus Quest 3 [15], Nreal's AR smartglass [17], Snapchat's Spectacles [22], and Apple's newly announced Vision Pro headset [3] are transforming previous visions for AR into market-ready products. In addition to the hardware advancements, the platforms and application development ecosystems surrounding these technologies are growing, as well. In this paper, we study five leading AR platforms: Apple's ARKit [9], Google's ARCore [5], Meta's Oculus Integration [20], Microsoft's MRTK [30] for Hololens, and WebXR [28] for the web. These platforms cover all three AR hardware form factors currently available: handheld mobile devices, video passthrough AR headsets, and optical see-through AR headsets.

UI in Augmented Reality. Unlike traditional 2D contexts where users interact with UI content on flat screens, the AR UI is a conjunction of visual elements from the physical world, the AR virtual world, and the user's interactions within the immersive 3D world. The AR context requires that the system process and understand the physical world surrounding the user [7,8,14,18,47], and virtual content is often "anchored" to physical-world surfaces, requiring rapid updates in response to change's in the user's physical surroundings or position.

Figure 1 shows how these elements are combined by platform-specific SDKs that process sensory data from the physical world as well as user input, and then pass this data to the platform's rendering engine to generate the final UI that is perceived by the user. Some SDKs include their own rendering engine; for example, both ARCore and ARKit handle rendering using the platform-specific built-in engine. Other SDKs offload rendering to external engines or libraries; e.g., while both Hololens 2 and Meta Quest provide their own platform SDKs, rendering is managed by existing game engines like Unity [25] or Unreal [26], while WebXR uses existing 3D rendering JavaScript libraries, such as Three.js [23] or Babylon.js [10]. The breadth and complexity of this ecosystem mean that different AR platforms may make different design or implementation choices in building the AR UI experience, and platform developers may fail to realize the security, privacy, and safety implications of these decisions or their integrations of different components.

Security and Privacy for AR. We add to a growing body of work from the computer security and privacy community, which has been addressing security, privacy, and safety risks in AR for over a decade [78]. The initial security threat modeling taxonomies for AR were proposed by Roesner et al., identifying input, data access, and output as key areas of concern [79]. Guzman et al. then built on these categories, incorporating user interaction and device protection [46]. Much prior AR security work falls into these taxonomies, including studies focusing on sensor data input privacy in AR/VR platforms [50, 55, 56, 86, 93, 95, 101], on device and network safety [52, 87, 90], on user input [63, 88] and work identifying and addressing malicious AR output [32, 43, 61]. Prior works also proposed mitigation strategies via AR platform design [56, 59, 61, 80, 81, 96]. Though some of these prior works touched on topics related to AR UI security (e.g., virtual content "output attacks" by malicious AR apps [43, 59]), to our knowledge no prior work has systematically studied AR UI security — including both application output and user input — as we do in this paper.

3 Selected Properties and Evaluation Metrics

In Section 1, we described five proof-of-concept attacks on AR UI security. Before we return to these attacks in Section 4, we first identify and introduce three key AR UI security related properties that underpin these types of attacks.

Properties. Our team, which included eight researchers, conducted multiple rounds of interactive threat modeling, structured brainstorming, and preliminary experiments to generate

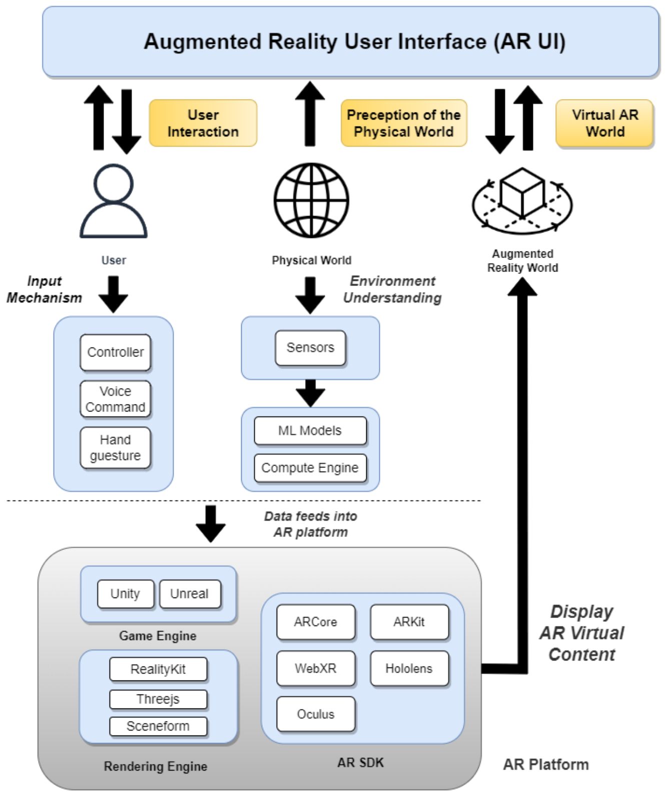


Figure 1: Overview of Augmented Reality User Interface. The yellow box represents the three main components: perception of the physical world, virtual AR world, and user interaction.

and refine ideas for design choices, properties, and/or test cases for AR platforms. Six (of eight) authors participated in the brainstorming process. This process involved: (1) each author independently generating security or privacy related questions and associated testable properties about AR platform designs (using sticky notes and a spreadsheet), (2) multiple authors reviewing and refining each row of the spreadsheet, and (3) clustering the properties according to themes (e.g., user input, multiple applications or components, hardware, sensors). From there, we chose to focus on AR UI security issues because we found them particularly interesting given the relationship between the UI, the environment, and the user and underexplored in current platforms.

1. **Same Space.** How do AR systems manage objects that share the same physical world mapping? For instance, when two AR objects with identical shapes and sizes are anchored at the same 3D coordinates, which object(s) become visible to the user? Which receive (s) the user's input? We leverage this property in the clickjacking attack (Section 4.2).
2. **Invisibility.** How do AR systems handle virtual objects in the AR world that are transparent? To what extent, if any, does an object's visibility influence its functionality?

For example, are transparent objects capable of receiving user input? What happens when a transparent object renders over another virtual object? We leverage this property in the user input denial-of-service (Section 4.3), object-in-the-middle (Section 4.5), and object erasure (Section 4.6) attacks.

3. **Synthetic User Input.** How do AR systems handle synthetic user input? For example, can adversarial code generate synthetic input to mimic human interaction, such as via a programmatically generated raycast? We leverage this property in the input forgery (Section 4.4) and the object-in-the-middle (Section 4.5) attacks.

Evaluation Metrics. Moving from these properties and questions to *specific evaluation metrics* that we can use in our empirical investigation of platforms in Section 5:

For **Same-Space,** if two virtual objects from different application components are created with the same size and placed at the same 3D coordinates, we evaluate:

- *Rendering Order:* Is the object placed first or the object placed second visible?
- *Interaction Order:* Does the object placed first or the object placed second receive user input?
- *Rendering Flicker-Free:* Is rendering order consistent within a single trial, or does it flicker?
- *Rendering Consistency:* Is rendering order consistent across trials?
- *Interaction Consistency:* Is interaction order consistent across trials?
- *Rendering-Interaction Consistency:* Are the object that is visible and the object that receive user input the same object?

For **Invisibility**, if objects are transparent, we evaluate:

- *Create Invisible Object:* Using each of the following possible invisibility mechanisms, can an invisible object be created? Mechanisms include: (1) setting a zero alpha value, (2) disabling the object's renderer, (3) using a null material for the object, and (4) using a custom transparent material for the object?
- *Invisible User Interaction:* Can invisible objects based on the preceding mechanism take user input?
- *Composes as Expected with Virtual Objects:* If an invisible object is rendered overlapping another virtual object, what is the resulting visible rendering? Is the other virtual object visible? [2]

For **Synthetic Input**, if input is not created by real AR users, we evaluate:

- *Create Synthetic Input:* Does the platform support synthetic user input, such as through a simulated raycast?
- *Invisible Synthetic Input:* When synthetic input is dispatched, has any visible indication to the user occurred (e.g., via a visible raycast)?
- *Input Provenance:* When a virtual object receives user input, is there a way for it to distinguish real user input from synthetic input?

We stress that completeness, in properties or metrics, is not our goal. Instead, we focus on metrics derived from our brainstorming activity that we considered important, challenging, and interesting in the AR context.

Valid Use Cases for Properties. While we focus on the security implication of these properties, we emphasize that they can also enable necessary features in many AR applications. Thus, seemingly simple solutions such as disallowing objects from occupying the same space, disabling invisible objects, or disallowing synthetic input will not be tenable.

The **Same Space** property allows designers and developers greater flexibility when creating AR experiences, enabling them to design complex scenes and arrangements where objects can interact, stack, or blend with each other in creative ways. In architectural or interior design AR applications, the ability for virtual objects to overlap becomes particularly useful as they can represent multiple layers or illustrate the relationships between different elements, providing a rich visual representation of the design. Moreover, the Same Space property facilitates intuitive user interaction and manipulation. Users can freely move around in the physical space, effortlessly viewing, moving, rotating, or scaling individual objects. The dynamic updating of visuals in response to changes in the user's physical surroundings or interaction occurs smoothly without rigid collision constraints, ensuring a seamless and natural experience.

The **Invisibility** property also contributes to a broad range of practical functionalities. For example, Pokemon Go, one of the most popular AR games, uses invisible objects as a placeholder for Pokemon that are still under development [31]. Moreover, existing research corroborates the value of this functionality in medical training scenarios [39]. Here, the AR objects are designed to be context-sensitive and can transition to an invisible state, allowing surgeons to look through overlaid content without obstruction. In addition, invisibility is also widely used for handling occlusion between virtual objects and real-world objects to ensure that virtual objects are realistically occluded by real-world objects, improving overall immersiveness [6].

The **Synthetic User Input** property, commonly implemented via raycasting, is integral to the functioning of AR features. Its primary function is to indicate user interaction and selection in the real world by returning information about the selection, such as the distance, position, or a reference to a real-world object (plane, surface) or virtual AR object. The many legitimate use cases of synthetic user input include, for example, an AR shooting game that generates synthetic input

[2]We were motivated to add this evaluation metric after discovering the object erasure attack for WebXR, described in Section 4.6.

as a shooting ray to intersect with the designated object.

Hence, it is crucial not to simply disable the features associated with these properties solely due to their potential security implications. Rather, we advocate that AR platform and application designers carefully consider the potential security implications in addition to the desirable use cases.

4 Threat Model and Proof-of-Concept Attacks

We now return to the proof-of-concept AR UI security attacks that we previewed in Section 1. First, we detail our threat model. Then, we describe in detail our proof-of-concept attacks on five AR platforms (ARCore, ARKit, Hololens, Oculus, and WebXR). Though we choose one platform on which to implement each attack, our empirical investigation in Section 5 will reveal that multiple platforms provide the preconditions for implementing most attacks.

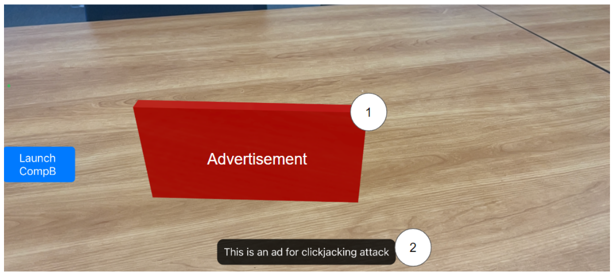
4.1 Threat Model And Attack Preconditions

We assume adversarial behavior might extend from one entity (e.g., an included ad library) to another entity (e.g., the main app), or vice versa. Attackers could directly call the API from the SDK to either gather sensitive information from the AR application (e.g., location of a target AR object) or place content within the AR virtual world. Our threat model resembles those used in other platforms, e.g., malicious third-party iframes [36,98] or included third-party libraries in the mobile ecosystem [97,100]. However, the current AR environment is arguably more vulnerable given there are no iframe-like primitives that isolate the execution of third-party code; instead, it shares a portion of the displayed AR scene.

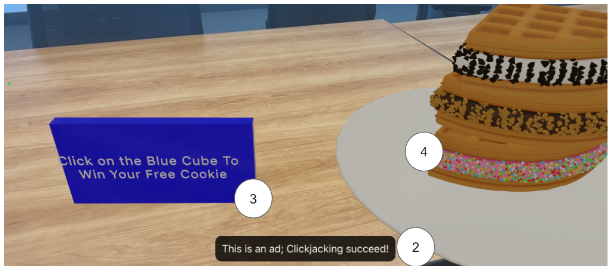
All proof-of-concept attacks rely on at most these three preconditions (in addition to the individual attack-specific metrics we test): (1) the location of the targeted object, (2) the ability to generate virtual content interleaved with the victim's content, and (3) the execution of synthetic input. These preconditions are straightforwardly met in today's systems, where third-party libraries are included in applications as either the attacker or victim components.

4.2 Clickjacking: Leveraging Inconsistency between Rendering and Interaction Orders

Attack Motivation: Bait User Interaction. Clickjacking is an attack that fools users into thinking they are clicking on one thing when they are actually clicking on another. As they can on web and mobile platforms today, app developers will be able to use third-party ad libraries on AR to display ads and generate revenue. AR advertising has attracted huge interest, and revenue in this market is projected to reach US \$1.05 billion in 2023 [4]. Attackers are well-incentivized to mount clickjacking attacks on ads it includes, tricking users into clicking on them and thereby increasing ad revenues. Prior work has hypothesized and demonstrated clickjacking attacks through hijacking the cursor in the context of VR [63]. Here, we demonstrate one type of clickjacking attack in AR without modifying the user's interaction.



(a) User's view of an AR advertisement object.


Click on the Blue Cube To Win Your Free Cookie
3
4
This is an ad; Clickjacking succeed!
2

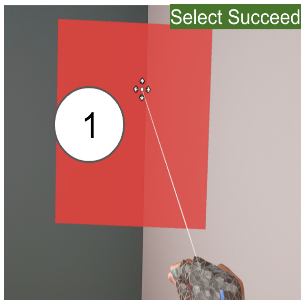
(b) Demonstration of the clickjacking attack

Figure 2: **Clickjacking attack on ARKit.** ① The advertisement object displayed in the AR world. ② A prompt after the advertisement is clicked. ③ A bait AR object rendered on top of the advertisement object exploiting the Same Space property. ④ An accompanying bait AR object. After the user clicks on the bait object ③, the interaction goes into the advertisement object ① and generates the prompt ②.

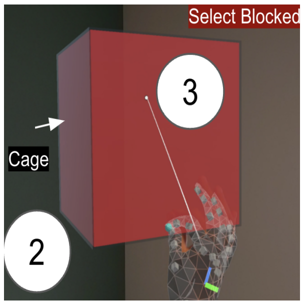
Attack Design. We implement this attack in iPhone 13 using the ARKit SDK. Our intuition is that the AR platform will handle the rendering sequence and interaction sequence differently when two objects are placed in the same 3D coordinates. For example, when we place two AR objects in ARKit using the same anchor ((`object.setParent(AnchorEntity)`)), the object placed second will always render over the first object, but the user's interaction will always trigger the functionality of the first, now hidden object. The attacker here is a revenue-hungry developer. The ad revenue will go to the developer as the user's interaction with the bait object goes into the ad object. The preconditions for this attack are (1) the location of the targeted object, and (2) the ability to generate virtual content interleaved with the victim's content.

Figure 2 illustrates the attack. When the victim/user launches the application, an advertising platform places a third-party ad ① in a certain bounded region of the main app, and a revenue-hungry developer/attacker then places a new interactive bait object ③ in the same space as the advertisement. This component displays the message "click here to win your free cookie" to bait user clicks. However, the user's interaction with the bait object actually triggers the underlying ad ① even as the attacker 'steals' revenue from the click.

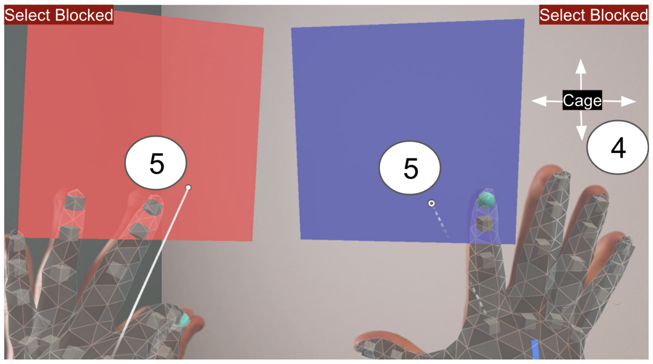
4.3 Denial-of-Service Proof-of-Concept: Leverage Invisibility



(a) User's view of the victim object. User is able to interact with the object.



(b) Demonstration of the first denial of service attack. "Cage" covers the object.

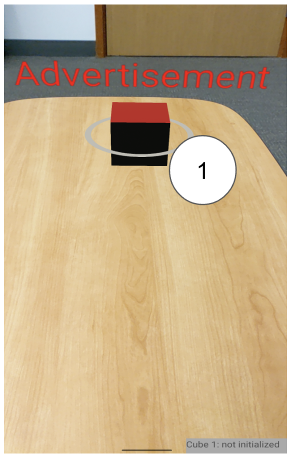


(c) Demonstration of the second denial of service attack. "Cage" covers the entire space.

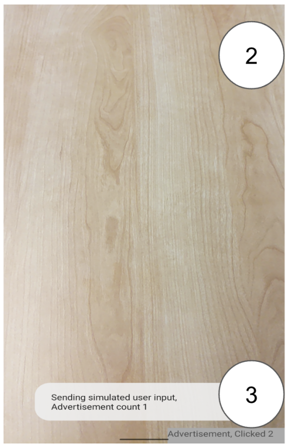
Figure 3: **Denial-of-user-input attack on Hololens.** ① The user can select and interact with the victim object (red). ② The attacker overlays a fully transparent object over the victim object (red). For demonstration, we make the transparent object ("cage") slightly visible. ③ The invisible "cage" blocks user interaction with the victim object (red). ④ In addition, the attacker can surround the user in a fully transparent "cage". ⑤ The invisible "cage" blocks the user from interacting with *any* AR objects , in this case, the red and blue AR objects.

Attack Motivation: Block User Interaction. A denial-of-service attack refers to an explicit attempt by a malicious entity to deny legitimate users access to an object/service. In this context, the attacker's goal is to stealthily prevent the user from interacting with the target AR object, e.g., preventing the user from engaging with a competitor's content or disrupting the user from properly engaging with *any* AR object.

Attack Design. We implement two variant of the denial-of-service attacks in Hololens 2 using the MRTK. The



(a) User's view of the victim app.



(b) Demonstration of the input forgery attack

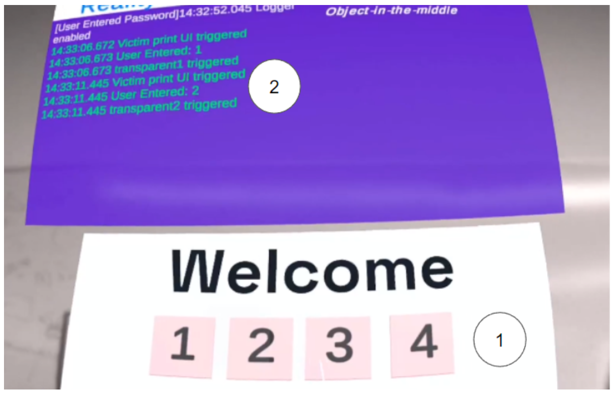
Figure 4: **Input forgery attack on ARCore.** ① The advertisement object displayed in the AR world. ② When the advertisement object is outside of the user's view, ③ the attacker launches a synthetic input to interact with the object. The prompt demonstrates the attack is successful.

attack insight is that the Hololens default raycast implementation, i.e., `private static RaycastResult PrioritizeRaycastResult`, would return the closest hit object. Based on our findings in Section 5, we find that an attacker can construct a completely invisible object that captures the user's input. Furthermore, the 3D nature of the AR virtual world lets the attacker envelop the user within this invisible 'object cage,' causing the invisible object to intercept, initially and always, the user's interactions. The preconditions for this attack are (1) the location of the targeted object, and (2) the ability to generate virtual content interleaved with the victim's content.

Figure 3 illustrates this attack. The victim/user launches the application and intends to select an AR object ① using a hand ray as the input mechanism. However, the attacker can either overlay an invisible object ② on the targetted AR object to block user from interacting with it ③. Furthermore, the attacker can overlay a large invisible object ④ that encapsulates the user such that the user is always physically inside of the object. The user's interaction towards *any* AR object is thus blocked ⑤ regardless of their movements within the physical space.

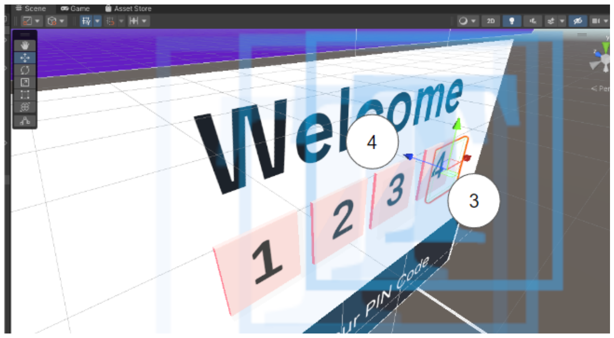
4.4 Input Forgery: Leveraging Synthetic User Input

Attack Motivation: Impersonate User Interaction. Similar to the motivation in the clickjacking attack, here the adversary's goal is to maximize advertisement engagement. The attacker places an advertisement object outside of the user's



```
[User Entered Password]14:32:52.045 Logger
enabled
14:33:06.672 Victim print UI triggered
14:33:06.673 User Entered: 1
14:33:06.673 transparent1 triggered
14:33:11.445 Victim print UI triggered
14:33:11.445 User Entered: 2
14:33:11.445 transparent2 triggered
```

(a) User's view of the victim app.



(b) Demonstration of the object-in-the-middle attack.

Figure 5: **Object-in-the-middle attack on Oculus.** ① The interface for authentication. ② The logger for the execution result. ③ The invisible object the attacker places over the pin pad object. ④ The blue arrow suggests the direction of the synthetic input to trigger the pin pad object.

view and generates synthetic user input to increase the number of ad interactions. Placing it outside of the user's peripheral is not a requirement though it will make the attack more stealthy.

Attack Design. We implement this attack in Pixel 5 using the ARCore SDK. The attack insight here is that a programmable click can interact with objects outside of user's view by exploring the limited display of field-of-view. The precondition for this attack is the execution of synthetic input.

Figure 4 illustrates the attack. When the victim/user launches the application, the malicious application developer will place the third-party ad ① in the user's view. When the ads' location is outside of the user's view ②, the attacker will then generate synthetic input ③ to trigger interactions on the ads, increasing its interaction count, and later charging the respective advertisers for this inflated number of ad views.

4.5 Intercepting User Inputs: Combining Invisible Objects and Synthetic User Input

Attack Motivation: Hijack User's Interaction. Object-in-the-middle is an attack in which a third party object gains access to (or "intercepts") the communication between two other objects. As AR applications continue to grow in domains beyond entertainment, users may enter sensitive information—such as PIN codes, passwords, or private messages—by interacting with a virtual keyboard rendered in the AR space. Each virtual key is a `collider` that provides collision detection. When the user either presses a key or casts a ray, the collider detects the intersection and outputs the corresponding value.

Prior work has shown the possibility to sniff user text input through various side-channels [69, 86]. Here, we demonstrate in AR that not only can an attacker surreptitiously steal the user's input, but it can also impersonate the user or even modify the original input, similar to a man-in-the-middle attack.

Attack Design. We implement this attack in the passthrough mode of Oculus Quest 2. Our attack intuitions here are twofold. First, similar to the denial-of-service attack, it is possible to block the user's original input by overlaying a transparent object on top of the keyboard. Second, we can cast a synthetic ray to impersonate the user's interaction since the current platform does not provide input provenance.

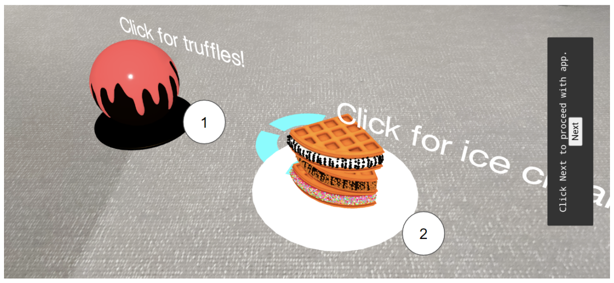
Given our observations, we implemented an object-in-the-middle attack on Oculus. The preconditions for this attack are (1) the location of the targeted object, (2) the ability to generate virtual content interleaved with the victim's content, and (3) the execution of synthetic input.

Figure 5 illustrates the attack. When the victim/user launches the application, it shows a mock authentication interface ①, part of the main app that consists of a pin pad for the user to enter their password. We present a logger ② to illustrate the efficacy of the attack. A malicious third-party component places several transparent meshes in front of the pin pad entity to intercept the user's input ③. Once the transparent meshes detect the user's interaction, the malicious component casts a synthetic input ④ to the pin pad, passing the user's input on to its intended destination.

The result is that the attacker has intercepted the password, but the user has not noticed anything amiss. Following the same attack logic, the attacker could also modify the user's input if needed since they can arbitrarily define the synthetic input destination.

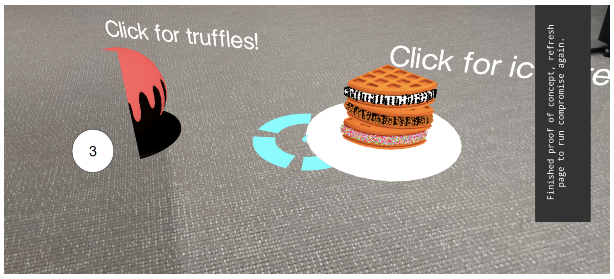
4.6 Object Erasure: Leveraging Invisible Meshes

Attack Motivation: Erase Other AR Objects. Echoing our attack focus, here we consider a scenario where the attacker aims to interfere with another party's virtual content by ma-



Click Next to proceed with app.

(a) User's view of victim app.



Finished proof of concept, refresh page to run compromise again.

(b) Demonstration of the object erasure attack.

Figure 6: **Object erasure attack on WebXR.** (①,② Two competing advertisements. ③ Attempt to erase the competitor advertisement using transparent mesh. (We partially offset the invisible object for attack visibility.

nipulating targeted objects. Motivations for such actions can vary greatly. For instance, one potential objective could be to erase a competitor's AR content, such as advertisements or promotional material. Another might be the desire to manipulate the user's perception: by altering or erasing certain AR objects, e.g., vital information or safety warnings, an attacker could significantly distort the user's view of reality [43, 61]. Such manipulation of AR content could also be deployed as a form of digital vandalism [57], comparable to defacing a physical sign or billboard.

Attack Design. We implement this attack in the Chrome browser using the WebXR SDK and Three.js library. When we assign a fully transparent mesh (png format) as the material of an AR object using `new THREE.MeshBasicMaterial( map: THREE.TextureLoader().load('transparent.png' ))`, we find that not only does the AR object become fully invisible, but it also "erases" the rendering of other AR objects positioned behind it. The preconditions for this attack are (1) the location of the targeted object and (2) the ability to generate virtual content interleaved with the victim's content.

Figure 6 illustrates the attack. When the user launches the application, it presents two competing third-party ad libraries ① and ②. Code from one mock ad library then places a transparent mesh in the same space as the competing advertisement to erase it ③.

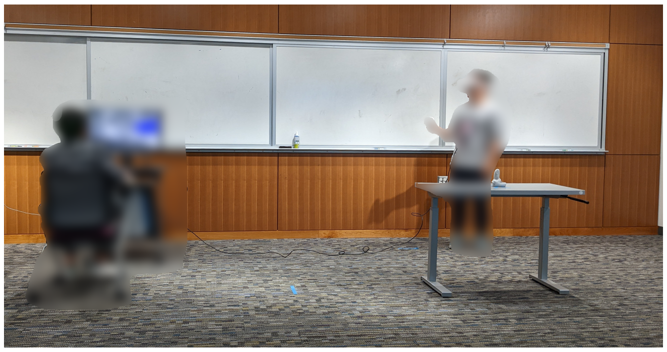


Figure 7: Researchers conducting the experiments.

5 Empirical Investigation

Stepping back from individual proof-of-concept attacks, we now turn to a more systematic investigation of current AR platforms. We evaluate how five leading AR platforms — ARCore (Google), ARKit (Apple), Oculus (Meta), Hololens (Microsoft), and WebXR (browser) — implement the AR UI properties identified in Section 3. In this section, we first describe the infrastructure for our experiments, detail the platforms and configurations we used, and then present results of experiments. Finally, we discuss the security implications of our results (Section 5.4).

5.1 Overall Experiment Infrastructure

We systematically developed a repeatable set of experiments for each property that we identified in the previous section (i.e., Same Space, Invisibility, and Synthetic Input) . The experiments we conducted varied depending on the evaluation metrics for that property.

A human experimenter ran each experiment and another researcher recorded the experiment results, as demonstrated in Figure 7. Each experiment follows the procedure described in Figure 8. The central component of each experiment is a harness launched at the procedure's beginning. Corresponding to our threat model, each experiment further involves two AR application components, ComponentA and ComponentB, representing two pieces of code from different entities (e.g., a main app and an embedded third-party library).

With the experimenter's input, the harness (1) launches ComponentA, then (after the experimenter made necessary observations) (2) launches ComponentB, and finally (3) steps through the experiment in four different locations, which we annotated as Location N, S, E, W, interacting with the AR object in different spatial locations and providing input to the harness to proceed.

To account for potential non-determinism, we conducted each test case experiment $M = 5$ times. Further, each test can have up to $N = 5$ sub-experiments. Details about ComponentA and ComponentB depend on the nature of each test case.

```
Experimenter's Procedure($M$,$N$)
    For $j$ = 1 to $M$ do
        Launch Harness($N$)
        Launch and provided any user input to start *experiment*
        Launch and provided any user input to start
            ComponentA.launch()
        Launch and provided any user input to start
            ComponentB.launch()
        For $k$ = 1 to $N$ do
            click "Next" button
            Start ComponentA.next()
            Start ComponentB.next()
            Perform sub-experiment observations and interactions
        Exit Harness($N$)
```

Figure 8: The parameter M denotes the number of experiments that the experimenter (a person) should run. Throughout the procedure's pseudocode is the assumption that the experimenter records detailed notes of observations during each experiment.

5.2 Experiment Platforms and Configurations

We describe here the specific experimental configurations and versions we used for each platform. An experimental setup requires not only a choice of platform (e.g., ARKit) but also additional choices for the entire tech stack (e.g., which SDK to use). Different choices for the full tech stack might have led to different experimental results, and we will release our experiment code to allow future work to adapt it for other contexts — including future technologies, like Apple's recently announced Vision Pro headset [3].

ARCore. ARCore [5] is Google's Android core SDK for augmented reality. We built our test cases using ARCore v1.32.0 [5] for AR functionalities and Sceneform SDK [12] for 3D content rendering (version 1.20.5). The test cases were implemented in Java and tested on the Pixel 5a.

ARKit. ARKit [9] is Apple's main AR SDK and includes multiple iOS AR frameworks, such as RealityKit [1] and RoomPlan [2]. We built our test cases using ARKit and RealityKit for necessary AR functionalities. The three test cases were implemented mainly in Swift and tested on the iPhone 13 Pro.

Oculus. We use Quest 2 with the experimental Passthrough API [40] and spatial anchors to enable an AR experience. We built our test cases using Oculus Integration SDK v44.0 [19]. They were implemented mainly in C# (Unity) and tested on the Oculus Quest 2.

WebXR. The WebXR Device API [28], led by the W3C Immersive Web Community Group, provides a uniform abstraction layer to host immersive web content. We built our test cases using the WebXR API-20220601 [29] for necessary AR functionalities and Three.js [23] for 3D content rendering. The three test cases were implemented mainly in JavaScript and tested on the Chrome Browser.

Hololens. We deployed a Hololens 2 application using the Mixed Reality Toolkit (MRTK) 2.0 [30] and Unity 2021.3.16f1 [24]. MRTK is an authorized Microsoft project that provides components to facilitate MR development in Unity. We employed numerous features of MRTK in our test cases, including spatial anchors, gesture recognition, and object manipulation.

5.3 Experiments and Results

We now examine our experiments and results, per property, based on the evaluation metrics presented in Section 3. Table 1 highlights all our results.

5.3.1 Same Space Experiments and Results

Experiment Design. In the SameSpace experiment, ComponentA creates a virtual Cube1 object at one coordinate in the physical world, and ComponentB creates a second Cube2 placed in the same physical location. Both Cubes use the same spatial anchor, which is a specific 6-degree-of-freedom pose (position and orientation) of the physical world, as the physical location reference and are registered with a user input handler. The experimenter moves to four different locations (noted as North (N), South (S), East (E), and West (W)). At each location, the experimenter observes the rendering sequence of the overlapping cubes and interacts with the Cubes by sending a virtual ray or tapping on the overlapping region to register the returning result. To verify our code is operating correctly, each Component displays additional cubes that the experimenter observes and verifies as part of the procedure. We omit further details about such checks because in all cases the checks were made and passed.

From these activities, the experimenter evaluates the metrics specified in Section 3. Specifically, the experimenter observes: which cube is visible (based on which color is visible), which cube takes user input (based on log output from the cube's input handler), and whether these observations are consistent across multiple conditions and trials.

Experiment Results. The top section of Table 1 presents the results. We find that platforms are inconsistent about which object is visible, which object receives input, and whether the visible object is the one receiving input. Significantly, these inconsistencies appear between platforms, across multiple trials of a single platform, and even within a single trial (i.e., flickering cubes). For example, for ARKit, Cube2 is always rendered, but Cube1 receives the user input; the opposite is true for HoloLens. For WebXR, there was significant inconsistency across multiple experiments as the overlapping Cube flickers and either cube could register the user's input. ARCore flickers in a slower way, but all of the user's input goes

Property	Condition	Metrics	ARCore	ARKit	Hololens	Oculus	WebXR
Same Space	Location N	Rendering Order	Unstable	②	①	Unstable*	Unstable
		Interaction Order	②	①	②	Unstable*	Unstable
	Location S	Rendering Order	Unstable	②	①	Unstable*	Unstable
		Interaction Order	②	①	②	Unstable*	Unstable
	Location E	Rendering Order	Unstable	②	①	Unstable*	Unstable
		Interaction Order	②	①	②	Unstable*	Unstable
	Location W	Rendering Order	Unstable	②	①	Unstable*	Unstable
		Interaction Order	②	①	②	Unstable*	Unstable
		Rendering Flicker-Free	○	●	●	●	○
		Rendering Consistency	○	●	●	○	○
		Interaction Consistency	●	●	●	○	○
		Rendering-Interaction Consistency	○	○	○	●	○
Invisibility	Alpha Value = 0	Create Invisible Object	○	●	○	●	○
		Invisible User Interaction	N/A	●	N/A	●	N/A
	Disable Renderer	Create Invisible Object	●	●	●	●	●
		Invisible User Interaction	○	○	●	●	●
	Customized Material	Create Invisible Object	●	○	○	○	●
		Invisible User Interaction	●	N/A	N/A	N/A	●
	Null Material	Create Invisible Object	○	○	○	○	●
		Invisible User Interaction	N/A	N/A	N/A	N/A	●
		Composes as Expected w/ Virtual Objects	●	●	●	●	○
Synthetic Input	Inside Field-of-view	Create Synthetic Input	●	●	●	●	●
		Invisible Synthetic Input	●	●	●	●	●
	Outside Field-of-view	Create Synthetic Input	●	●	●	●	●
		Invisible Synthetic Input	●	●	●	●	●
		Input Provenance	○	○	○	○	○

Table 1: Overview of experiment results. Filled circles indicate the metric is satisfied, and empty circles indicate they are not. For the Same Space experiment, circled 1 indicates that the object is created by the first (victim) principal; circled 2 indicates that the object is created by the second (adversarial) principal. "Unstable" means that the results are inconsistent during a single trial. "Unstable*" means that the results are inconsistent between multiple trials.

into Cube2.

We find that Oculus is the only platform that handles rendering-interaction consistently, with a caveat that 15% (15 out of 100) of the trials fail to maintain this consistency. While Oculus prevents flickering from the overlapping Cube, we find that the rendering sequence and the corresponding interaction sequence flip as the user moves to the next testing location. In addition, we also notice an inconsistency across multiple trails in terms of which Cube appears on top.

These findings suggest that current AR platforms have not systematically considered the Same Space property or these types of corner cases.

5.3.2 Invisibility Experiments and Results

Experiment Design. In the Invisibilityexperiment, ComponentA creates a visible Cube1 object at one coordinate, and ComponentB creates a visible Cube2 in front of Cube1. Cube2 is significantly larger than Cube1. The harness asks the experimenter to confirm if Cube2 completely occludes Cube1 and, upon confirmation, ComponentB deletes Cube2 and creates a new invisible Cube2' at the same location and with the same geometry; it does this by attempting four different mechanisms: (1) setting the alpha value to zero, (2) providing a null material, (3) disabling rendering, and (4) uploading a transparent texture as the object's material. Both Cubes are registered with a user input handler. The process with Cube2 size and placement verification and then Cube2' instantiation ensures that input makes sure any interaction will go into Cube2' first. The experimenter then evaluates the metrics from Section 3, observing whether the invisible cube is fully invisible and dispatching user input to the invisible cube to observe if its input handler is triggered.

Experiment Results. The middle section of Table 1 presents the results. We find that all five AR platforms can create invisible objects that take the user's input. However, the detailed implementation condition for each platform differs slightly. For example, by setting Cube2's alpha value to zero, ARKit and Oculus both generate a completely transparent cube while registering the user's input. When disabling the rendering, ARCore and ARKit completely occlude Cube2', meaning that the cube can neither be seen nor receive input; Cube2' in WebXR, Hololens, and Oculus instead take input while being fully invisible. For Hololens and Oculus, this occurs because the rendering and collision engines are separate, which means objects can still possess physical characteristics without being rendered.

When uploading the customized transparent texture, we find that ARCore and WebXR produce a fully invisible Cube2 that takes input, while the other three platforms generate a slightly visible Cube2. All platforms except WebXR handle the null material edge cases by generating a solid color Cube2. Surprisingly, we found that the AR object generated by one principal in WebXR can affect the appearance of *other* AR objects. Specifically, whenCube2' is created with an uploaded transparent texture, we observe that the visible object located behind the invisible object (based on the user's viewing position) seems to disappear.

5.3.3 Synthetic Input Experiments and Results

Experiment Design. In the Synthetic Input experiment, ComponentA creates a virtual Cube1 object at one coordinate with an input handler. When the experimenter presses a button, ComponentB creates synthetic user input — in the form of a simulated raycast — to interact in the direction of Cube1. The direction of the simulated raycast is calculated by retrieving the location of Cube1 and generating the corresponding direction vector. In addition, we test if input visualization is required to generate synthetic input. The experimenter observes whether the input handler of Cube1 was triggered. This experiment is repeated with the target object (Cube1) inside the user's field of view and outside the user's field of view (e.g., behind the user).

Experiment Results. The bottom section of Table 1 presents the results. We find that all platforms allow synthetic user input, which is moreover invisible *and* can interact with the target AR object (Cube1). One consistent observation across all platforms is the absence of effective input provenance verification. This is a significant shortfall as the raycast API does not supply sufficient information to distinguish between genuine and synthetic user input. In addition, we discovered that for synthetic user input to be functional, the target objects need not be within the user's field of view. This leverages users' limited visual awareness when they are physically present within the 360-degree immersive AR user interface when the AR objects themselves remain rendered within the scene, exploiting this gap in user perception.

5.4 Security Implications

Returning to our proof-of-concept attacks from Section 4, we can now directly connect our empirical evaluation results with those attacks. In Table 2, we connect each attack to its preconditions — that is, to the necessary experiment result(s) that would enable the attack. For example, our denial-of-service attack requires that a platform be able to create invisible cubes and that those cubes be able to receive user input (i.e., "Invisible Cube Input Registration: ●"). Based on our experiment results in Table 1 and each of the attack preconditions, we can thus summarize in Table 2 which of our tested SDKs are vulnerable to which attack. We demonstrate all five proof-of-concept attacks in current AR platforms in Section 4.

Proposed Attack	Property	Attack Precondition	ARCore	ARKit	Hololens	Oculus	WebXR
Denial-of-service	Invisibility Invisibility	Create Invisible Object: ● Invisible User Interaction: ●	★	★	★	★	★
Object Erasure	Invisibility	Composes as Expected w/ Virtual Obj.: ○					★
Input Forgery	Synthetic Input	Input Provenance: ○	★	★	★	★	★
Clickjacking	Same Space Same Space	Interaction Consistency: ● Rendering-Interaction Consistency: ○		★	★	★*	
Object-in-the- Middle	Invisibility Synthetic Input Synthetic Input	Invisible Interaction: ● Input Provenance: ○ Invisible Synthetic Input: ●	★	★	★	★	★

Table 2: Analysis of which attacks each platform (in our testing configuration) enables. ★ indicates this attack is possible to implement based on our experimental results; * indicates the attack might fail due to the AR platform's inconsistent behavior.

6 Discussion

As with all emerging technologies — from smartphones (in the early 2000s) to computerized automobiles (also in the early 2000s) — many defensive strategies can be adopted from earlier technologies and new challenges must be overcome. We view this paper as one component in the evolution of computer security and privacy for AR systems. While it is impossible to predict the future, we reflect on our work's possible place in this evolution here.

6.1 Problem Formulation

There are often vulnerabilities in emerging technologies that, from a purely abstract technical perspective, are unsurprising. Consider, for example, the discovery of `strcpy` vulnerabilities in automobiles in 2011 [41], over two decades after Aleph One's classic "Smashing the Stack for Fun and Profit" [72]. The contribution of such results is not in finding yet another vulnerability but in assessing *how* an emerging technology — one that has not yet seen significant security analysis — *might* be vulnerable. We consider our work — our identification of properties to assess as well as our exploration of case study attacks — to be of this lineage. Moreover, we consider not only how AR systems might be vulnerable, but we empirically find that the designs of *all* five instantiations of the AR technologies that we study *do* expose themselves to attacks.

Today's AR systems may not need to be secure against the types of attacks we study. They are still emerging and predominantly single-principal. But, under the assumptions that future systems might be multi-principal, more widely used, and make design decisions that often persist in future technologies even as the threat landscape changes, we believe that it is imperative for AR platform designers to consider mitigation strategies now.

6.2 Knowledge of Defenses for 2D UIs Will Help

Applying existing (known) defensive strategies from other domains is a natural and appropriate first line of defense when considering newly discovered vulnerabilities in emerging technologies. For example, in the automotive domain, early research suggested the use of (even then) standard defenses, such as application-level authentication and encryption and the avoidance of unsafe code like `strcpy` [41]. Likewise, there is already an existing wealth of knowledge on UI security for 2D interfaces (desktop, mobile, web), and we encourage the adoption (or extension) of those defensive techniques to AR systems.

For example, an invariant that can provide resilience to clickjacking on the web is the following: a user can interact with a web object if and only if the web object is visible and has been visible for at least a minimum amount of time [54]. That defensive strategy, if implemented and fully instantiated, would serve to strengthen AR systems. However, is it possible to fully instantiate that defensive strategy? We elaborate on this question below.

Other known techniques that could be adapted for AR UI security — once the need to adapt them has been identified, as through our work — include: (1) input provenance to help the application distinguish real user input from synthetic input, as well as synthetic input from different sources [11, 49]; (2) "Z-fighting" (same space) mitigations for same space conflicts (such as slightly offsetting multiple objects [21] or higher resolution buffers [58]); and (3) the isolation of different application components (i.e., a "same origin policy" for AR applications).

However, applying these techniques for AR may not always be straightforward or sufficient, as we elaborate in the next subsection.

6.3 Knowledge of Defenses for 2D UIs Is Not Sufficient

Though it is impossible to fully predict all the challenges with future 3D interfaces, we observe several potential differences here.

For example, consider known defenses for clickjacking. From our assessment of 3D gaming environments, there *are* reasons for which developers might intentionally create invisible yet interactable objects. Thus, the anti-clickjacking invariant mentioned above may not be directly applicable in all immersive 3D environments and use cases. Further, unlike desktop and web environments, where one object is clearly "on top," the visibility of an object might be impacted by the position of the viewer, who could be moving around the objects, or the objects could be moving around the viewer. This "on top" nature becomes even more complicated if the same set of objects are being viewed by multiple people (different viewers of the same scene might see different objects on top). Certainly, computational methods could be used to determine which object is on top of each user, though platform designers must account for the different architecture of AR systems and the different roles of the physics and rendering engines. Even then, some invariants, such as minimum time of visibility before being clickable, may be incompatible with some use cases, like playing a game with fast-moving objects while simultaneously using another leisure application.

As another example, consider that our attacks depend on a threat model in which multiple application components — or multiple applications — are mutually distrusting. While current AR platforms do not (yet) support rich multi-application interactions (except with applications confined to 2D windows), we can anticipate that future platforms will evolve to fully support two (or more) immersive augmented reality applications at the same time: for example, walking directions interspersed with game content from another application. Securely managing the integration of content from multiple applications in the AR context, or even isolating content from different sources within the same application (like Site Isolation for iframes on the web [75]), will be a large technical and research challenge [60].

6.4 Potential Defensive Techniques

Work has already begun to emerge within the security research community that provides potential defensive directions against the issues we raise in this paper. For example, Lee et al. [63] introduced AdCube, a defensive system to counter several WebXR attacks. The attacks they consider have some similarity (and some differences) with the ones we discuss here. The blind spot tracking attack from AdCube places the ad entity outside of the user's peripheral to increase the number of ad appearances, while the input forgery attack in this paper maximizes the number of ad clicks through synthetic input. The cursor-jacking attacks from AdCube exploit the user's perception to hijack authentic clicks whereas our clickjacking attack utilizes the rendering-interaction inconsistency of virtual objects placed in the same space. Despite these differences, the AdCube idea of confining untrustworthy third-party libraries and preventing them from placing virtual objects could mitigate attacks like those we explore as well: specifically, attacks that rely on the ability to generate virtual content interleaved with the victim's content. Future researchers could build upon AdCube and our findings to develop a cross-platform AR defensive toolkit. More generally, future work should also explore other possible approaches that isolate UI components from multiple principals in future AR platforms.

Another line of work that may contribute to defensive directions focused on automated evaluation in AR/VR. Wang et al. [94] defined and used interactable properties of virtual objects to perform automatic testing in the VR environment. Lehman et al. [65] provided automated feedback to help with UX issues debugging. Our methodology is inspired by property-based testing (PBT), a testing technique that leverages the specification of generic properties as the driving force behind the testing process, ensuring that the system under test satisfies these properties [51]. There is rich literature on applying PBT in programming languages [45, 53] and 2D rendering engines [68]. Building on top of our findings, future work could automatically generate test cases analyzing specific metrics — for example, rendering-interaction consistency across multiple conditions, since our results show that the platform may have inconsistent behavior in different scenarios.

6.5 Stepping Back

The preceding observations do not imply that the types of attacks we describe here are insurmountable. Rather, this discussion reflects our view of how this work might fit into the broader evolution of security and privacy for AR systems. At the highest level, and returning to Section 6.1, we believe that our first and most important contribution is the knowledge of *what* vulnerabilities might exist and how they presently manifest in different systems. With that knowledge, it is possible to defend systems by leveraging existing knowledge from 2D contexts (Section 6.2), by creating new knowledge (Section 6.3), and by extending existing defensive directions from AR contexts (Section 6.4). Toward the creation of new knowledge and new defenses, a critical step will be the assessment of what types of applications and use cases the designers intend to support and the management of tensions that arise when the exact feature exploited by an adversarial UI application is also needed by a desirable application.

7 Additional Related Work

Finally, we connect to additional related work in the broader context of AR platform analysis and 2D UI security that was not previously discussed.

Empirical Analysis of AR Platforms. As AR platforms continue to emerge and develop, recent work has compared and evaluated AR platforms according to other criteria as well. Scargill et al. [83] investigate AR object placement stability in mobile AR platforms. Slocum et al. [85] measure the spatial inconsistency when placing virtual objects in the real world on ARCore, while Lee et al. [62, 76] analyze the AR object placement deviation on WebXR. Other works have proposed functionality metrics, such as general performance (CPU/memory use) [71] body movement and marker-based tracking [34, 91], accessibility and ease-of-use [89], lighting estimation [73], and plane and feature point detection [73] that allow direct comparison across multiple AR platforms.

2D UI Security. UI security in 2D has been well-studied. For example, early work in this space included secure windowing systems like Trusted X [48] and EROS [84]. More recently, a line of work considered UI security requirements and threats on Android and iOS [35, 38, 42, 64, 77]. Luo et al. [66] provide a thorough analysis of UI vulnerabilities in mobile browsers. As mentioned, there is also significant prior work mitigating clickjacking attacks on the web and in other contexts (e.g., [54]). Our work takes the next step in the broader space of UI-level security, studying emerging AR platforms.

8 Conclusion

We presented an empirical analysis of five current AR platforms, systematically investigating how they handle three UI security related properties: Same Space, Invisibility, and Synthetic Input. We demonstrated five proof-of-concept attacks — one implemented for each of our test platforms — that leverage different design choices in the context of these AR UI security properties. We found that these current AR platforms, including Apple's ARKit, Google's ARCore, Meta's Oculus, Microsoft's Hololens, and WebXR, are all designed and implemented in ways that enable our AR UI attacks to succeed. Our findings lay the groundwork for future research and design work to consider and address AR UI security to mitigate the risks of such attacks, either through directly applying past lessons from 2D UI security or by grappling with new, AR-specific challenges and tradeoffs.

Along with this paper, all code for our experiments and the video demonstrations are available online at https://ar-sec.cs.washington.edu/ar_ui/ to support future researchers in extending our analyses to other and future AR platforms, including the Apple Vision Pro headset that was announced on the day before this paper's submission.

Acknowledgments

We would like to thank our shepherd and the anonymous reviewers for their valuable feedback. We would also like to thank the following people for their feedback: Michael Flanders, Alex Gantman, Gregor Haas, Rachel Hong, Sandy Kaplan, Inna Wanyin Lin, Rachel McAmis, Kentrell Owens, and Yuan Tian. This work was supported in part by the National Science Foundation under Award #1651230, as well as by awards from Cisco, Google, and Qualcomm.

References

[1] Apple RealityKit. https://developer.apple.com/documentation/realitykit/.

[2] Apple RoomPlan. https://developer.apple.com/augmented-reality/roomplan/.

[3] Apple Vision Pro is Apple's new AR headset. https://www.theverge.com/2023/6/5/23738968/apple-vision-pro-ar-headset-features-specs-price-release-date-wwdc-2023.

[4] AR Advertising Market Insights. https://www.statista.com/outlook/amo/ar-vr/ar-advertising/united-states.

[5] ARCore. https://developers.google.com/ar/develop.

[6] ARCore – Depth adds realism. https://developers.google.com/ar/develop/depth.

[7] ARCore – Geospatial. https://developers.google.com/ar/develop/geospatial.

[8] ARCore – Light Estimation. https://developers.google.com/ar/develop/lighting-estimation.

[9] ARKit. https://developer.apple.com/augmented-reality/arkit/.

[10] Babylon.js. https://www.babylonjs.com/.

[11] Event: isTrusted property. https://developer.mozilla.org/en-US/docs/Web/API/Event/isTrusted.

[12] Google Sceneform. https://developers.google.com/sceneform/develop.

[13] Headers/X-Frame-Options. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options.

[14] Hololens – Scene understanding. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/scene-understanding-SDK.

[15] Meta Quest 3. https://about.fb.com/news/2023/06/meta-quest-3-coming-this-fall/.

[16] Microsoft hololens 2. https://www.microsoft.com/en-us/hololens.

[17] Nreal Light AR Glasses. https://www.verizon.com/products/nreal-light-ar-glasses.

[18] Oculus – Spatial Anchor. https://developer.oculus.com/documentation/unity/unity-spatial-anchors-overview/.

[19] Oculus Integration with Unity. https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022.

[20] Oculus Passthrough. https://developer.oculus.com/documentation/unity/unity-passthrough/.

[21] Shaderlab command: Offset. https://docs.unity3d.com/2021.1/Documentation/Manual/SL-Offset.html.

[22] Spectacles from Snapchat. https://www.spectacles.com/.

[23] Three.js. https://threejs.org/.

[24] Unity 2021.3.16. https://unity.com/releases/editor/whats-new/2021.3.16l.

[25] Unity AR Foundation. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@2.2/manual/index.html.

[26] Unreal Game Engine. https://www.unrealengine.com/en-US.

[27] User Interface Security and the Visibility API – Privacy Consideration. https://www.w3.org/TR/UISecurity/#privacy-considerations.

[28] WebXR Device API. https://www.w3.org/TR/webxr/.

[29] WebXR Device API Version 20220601. https://www.w3.org/TR/2022/CRD-webxr-20220601/.

[30] What is Mixed Reality Toolkit 2? https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/.

[31] What Is The Invisible Obstacle In Pokémon Go? https://www.ginx.tv/en/pokemon-go/invisible-obstacle.

[32] Surin Ahn, Maria Gorlatova, Parinaz Naghizadeh, Mung Chiang, and Prateek Mittal. Adaptive fog-based output security for augmented reality. In *Proceedings of the Morning Workshop on Virtual Reality and Augmented Reality Network*, pages 1–6, 2018.

[33] Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, and Dawn Song. Clickjacking revisited: A perceptual view of UI security. In *8th USENIX Workshop on Offensive Technologies WOOT 14)*, 2014.

[34] Dhiraj Amin and Sharvari Govilkar. Comparative study of augmented reality SDKs. *International Journal on Computational Science & Applications*, 5(1):11–26, 2015.

[35] Simone Aonzo, Alessio Merlo, Giulio Tavella, and Yanick Fratantonio. Phishing attacks on modern android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1788–1801, 2018.

[36] Sajjad Arshad, Amin Kharraz, and William Robertson. Include me out: In-browser detection of malicious third-party content inclusions. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*, pages 441–459. Springer, 2017.

[37] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 135–144, 2010.

[38] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. What the app is that? deception and countermeasures in the android user interface. In *2015 IEEE Symposium on Security and Privacy*, pages 931–948. IEEE, 2015.

[39] Christoph Bichlmeier, Felix Wimmer, Sandro Michael Heining, and Nassir Navab. Contextual anatomic mimesis hybrid in-situ visualization method for improving multi-sensory depth perception in medical augmented reality. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 129–138. IEEE, 2007.

[40] Gaurav Chaurasia, Arthur Nieuwoudt, Alexandru-Eugen Ichim, Richard Szeliski, and Alexander Sorkine-Hornung. Passthrough+ real-time stereoscopic view synthesis for mobile mixed reality. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(1):1–17, 2020.

[41] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[42] Qi Alfred Chen, Zhiyun Qian, and Z Morley Mao. Peeking into your app without actually seeing it: UI state inference and novel android attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 1037–1052, 2014.

[43] Kaiming Cheng, Jeffery F Tian, Tadayoshi Kohno, and Franziska Roesner. Exploring user reactions and mental models towards perceptual manipulation attacks in mixed reality. In *USENIX Security*, volume 18, 2023.

[44] Kang Leng Chiew, Kelvin Sheng Chek Yong, and Choon Lin Tan. A survey of phishing attacks: Their types, vectors and technical approaches. *Expert Systems with Applications*, 106:1–20, 2018.

[45] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 268–279, 2000.

[46] Jaybie A De Guzman, Kanchana Thilakarathna, and Aruna Seneviratne. Security and privacy approaches in mixed reality: A literature survey. *ACM Computing Surveys (CSUR)*, 52(6):1–37, 2019.

[47] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, et al. DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 829–843, 2020.

[48] J. Epstein, J. McHugh, R. Pascale, C. Martin, D. Rothnie, H. Orman, A. Marmor-Squires, M. Branstad, and B. Danner.

Evolution of a trusted B3 window system prototype. In *IEEE Symposium on Security and Privacy*, 1992.

[49] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steve Hanna, and Erika Chin. Permission Re-Delegation: Attacks and defenses. In *USENIX Security Symposium*, 2011.

[50] Lucas Silva Figueiredo, Benjamin Livshits, David Molnar, and Margus Veanes. Prepose: Privacy, security, and reliability for gesture-based programming. In *IEEE Symposium on Security and Privacy (SP)*, pages 122–137, 2016.

[51] George Fink and Matt Bishop. Property-based testing: a new approach to testing for assurance. *ACM SIGSOFT Software Engineering howpublisheds*, 22(4):74–80, 1997.

[52] Jassim Happa, Mashhuda Glencross, and Anthony Steed. Cyber security threats and challenges in collaborative mixed-reality. *Frontiers in ICT*, 6:5, 2019.

[53] Eric Dean Haugh and Matt Bishop. *Testing C programs for buffer overflow vulnerabilities*. PhD thesis, Citeseer, 2002.

[54] Lin-Shung Huang, Alexander Moshchuk, Helen J Wang, Stuart Schechter, and Collin Jackson. Clickjacking: Attacks and defenses. In *USENIX security symposium*, pages 413–428, 2012.

[55] Suman Jana, David Molnar, Alexander Moshchuk, Alan Dunn, Benjamin Livshits, Helen J Wang, and Eyal Ofek. Enabling fine-grained permissions for augmented reality applications with recognizers. In *22nd USENIX Security Symposium*, pages 415–430, 2013.

[56] Suman Jana, Arvind Narayanan, and Vitaly Shmatikov. A scanner darkly: Protecting user privacy from perceptual applications. In *IEEE Symposium on Security and Privacy*, pages 349–363, 2013.

[57] Levente Juhász, Tessio Novack, Hartwig H Hochmair, and Sen Qiao. Cartographic vandalism in the era of location-based games—the case of openstreetmap and pokémon go. *ISPRS International Journal of Geo-Information*, 9(4):197, 2020.

[58] Mark Kilgard. Creating reflections and shadows using stencil buffers. In *At Game Developers Conference*, volume 7, 1999.

[59] Kiron Lebeck, Tadayoshi Kohno, and Franziska 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, 2016.

[60] Kiron Lebeck, Tadayoshi Kohno, and Franziska Roesner. Enabling multiple applications to simultaneously augment reality: Challenges and directions. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 81–86, 2019.

[61] Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Securing augmented reality output. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 320–337. IEEE, 2017.

[62] Daehyeon Lee, Woosung Shim, Munyong Lee, Seunghyun Lee, Kye-Dong Jung, and Soonchul Kwon. Performance Evaluation of Ground AR Anchor with WebXR Device API. *Applied Sciences*, 11(17):7877, 2021.

[63] Hyunjoo Lee, Jiyeon Lee, Daejun Kim, Suman Jana, Insik Shin, and Sooel Son. AdCube: WebVR Ad Fraud and Practical Confinement of Third-Party Ads. In *USENIX Security Symposium*, pages 2543–2560, 2021.

[64] Yeonjoon Lee, Xueqiang Wang, Kwangwuk Lee, Xiaojing Liao, XiaoFeng Wang, Tongxin Li, and Xianghang Mi. Understanding iOS-based Crowdturfing Through Hidden UI Analysis. In *USENIX Security Symposium*, pages 765–781, 2019.

[65] Sarah M Lehman, Haibin Ling, and Chiu C Tan. Archie: A user-focused framework for testing augmented reality applications in the wild. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 903–912. IEEE, 2020.

[66] Meng Luo, Oleksii Starov, Nima Honarmand, and Nick Nikiforakis. Hindsight: Understanding the evolution of UI vulnerabilities in mobile browsers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 149–162, 2017.

[67] Tongbo Luo, Xing Jin, Ajai Ananthanarayanan, and Wenliang Du. Touchjacking attacks on web in android, ios, and windows phone. In *Foundations and Practice of Security: 5th International Symposium, FPS 2012, Montreal, QC, Canada, October 25-26, 2012, Revised Selected Papers 5*, pages 227–243. Springer, 2013.

[68] Joel Martin and David Levine. Property-based testing of browser rendering engines with a consensus oracle. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 424–429. IEEE, 2018.

[69] Ülkü Meteriz-Yıldıran, Necip Fazıl Yıldıran, Amro Awad, and David Mohaisen. A keylogging inference attack on air-tapping keyboards in virtual environments. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 765–774. IEEE, 2022.

[70] Vivek Nair, Wenbo Guo, Justus Mattern, Rui Wang, James F O'Brien, Louis Rosenberg, and Dawn Song. Unique identification of 50,000+ virtual reality users from head & hand motion data. *arXiv preprint arXiv:2302.08927*, 2023.

[71] Paweł Nowacki and Marek Woda. Capabilities of ARcore and ARkit platforms for AR/VR applications. In *International Conference on Dependability and Complex Systems*, pages 358–370. Springer, 2019.

[72] Aleph One. Smashing the stack for fun and profit. *Phrack*, 7(49), November 1996.

[73] Zainab Oufqir, Abdellatif El Abderrahmani, and Khalid Satori. ARKit and ARCore in serve to augmented reality. In *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–7. IEEE, 2020.

[74] Joe Gibbs Politz, Spiridon Eliopoulos, Arjun Guha, and Shriram Krishnamurthi. Adsafety: Type-based verification of javascript sandboxing. *arXiv preprint arXiv:1506.07813*, 2015.

[75] Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *USENIX Security Symposium*, 2019.

[76] Olle Renius. A Technical Evaluation of the WebXR Device API for Developing Augmented Reality Web Applications, 2019.

[77] Franziska Roesner and Tadayoshi Kohno. Securing embedded user interfaces: Android and beyond. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 97–112, 2013.

[78] Franziska Roesner and Tadayoshi Kohno. Security and privacy for augmented reality: Our 10-year retrospective. In *VR4Sec: 1st International Workshop on Security for XR and XR for Security*, 2021.

[79] Franziska Roesner, Tadayoshi Kohno, and David Molnar. Security and privacy for augmented reality systems. *Communications of the ACM*, 57(4):88–96, 2014.

[80] Franziska Roesner, David Molnar, Alexander Moshchuk, Tadayoshi Kohno, and Helen J Wang. World-driven access control for continuous sensing. In *ACM Conference on Computer and Communications Security*, pages 1169–1181, 2014.

[81] Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Secure Multi-Sser content sharing for augmented reality applications. In *28th USENIX Security Symposium*, pages 141–158, 2019.

[82] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web*, 2(6):24, 2010.

[83] Tim Scargill, Gopika Premsankar, Jiasi Chen, and Maria Gorlatova. Here To Stay: A Quantitative Comparison of Virtual Object Stability in Markerless Mobile AR. In *Proc. IEEE/ACM Workshop on Cyber-Physical-Human System Design and Implementation*, 2022.

[84] Jonathan S. Shapiro, John Vanderburgh, Eric Northup, and David Chizmadia. Design of the EROS trusted window system. In *USENIX Security Symposium*, 2004.

[85] Carter Slocum, Xukan Ran, and Jiasi Chen. RealityCheck: A tool to evaluate spatial inconsistency in augmented reality. In *2021 IEEE International Symposium on Multimedia (ISM)*, pages 58–65. IEEE, 2021.

[86] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions:AR/VR keylogging from user head motions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 159–174, 2023.

[87] Ivo Sluganovic. *Security of mixed reality systems: authenticating users, devices, and data*. PhD thesis, University of Oxford, 2018.

[88] Zihao Su, Faysal Hossain Shezan, Yuan Tian, David Evans, and Seongkook Heo. Perception Hacking for 2D Cursorjacking in Virtual Reality. 2022.

[89] Porfirio Tramontana, Marco De Luca, and Anna Rita Fasolino. An Approach for Model Based Testing of Augmented Reality Applications. In *RCIS Workshops*, 2022.

[90] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR. In *31st USENIX security symposium (USENIX security 22)*, pages 3789–3806, 2022.

[91] Tetiana A Vakaliuk and Svitlana I Pochtoviuk. Analysis of tools for the development of augmented reality technologies. CEUR Workshop Proceedings, 2021.

[92] Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, and Wouter Joosen. Webjail: least-privilege integration of third-party components in web mashups. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 307–316, 2011.

[93] John Vilk, David Molnar, Benjamin Livshits, Eyal Ofek, Chris Rossbach, Alexander Moshchuk, Helen J Wang, and Ran Gal. SurroundWeb: Mitigating privacy concerns in a 3D web browser. In *IEEE Symposium on Security and Privacy*, pages 431–446, 2015.

[94] Xiaoyin Wang. Vrtest: an extensible framework for automatic testing of virtual reality scenes. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 232–236, 2022.

[95] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 3382–3398. IEEE Computer Society, 2023.

[96] Eisa Zarepour, Mohammadreza Hosseini, Salil S Kanhere, and Arcot Sowmya. A context-based privacy preserving framework for wearable visual lifeloggers. In *IEEE PerCom Workshops*, pages 1–4, 2016.

[97] Xian Zhan, Lingling Fan, Sen Chen, Feng We, Tianming Liu, Xiapu Luo, and Yang Liu. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1695–1707. IEEE, 2021.

[98] Mingxue Zhang, Wei Meng, Sangho Lee, Byoungyoung Lee, and Xinyu Xing. All your clicks belong to me: Investigating click interception on the web. In *USENIX Security Symposium*, pages 941–957, 2019.

[99] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. It's all in your head (set): Side-channel attacks on ar/vr systems. In *USENIX Security*, 2023.

[100] Zicheng Zhang, Wenrui Diao, Chengyu Hu, Shanqing Guo, Chaoshun Zuo, and Li Li. An empirical study of potentially malicious third-party libraries in android apps. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 144–154, 2020.

[101] Yiqin Zhao, Sheng Wei, and Tian Guo. Privacy-preserving reflection rendering for augmented reality. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 2909–2918, 2022.