ProViewer: A Proximity- and View-Aware Framework for Detecting UI Security Risks in Extended Reality Applications

Bertram Liu
George Mason University
Fairfax, VA, USA
bliu22@gmu.edu

Vamsi Shankar Simhadri
George Mason University
Fairfax, VA, USA
vsimhadr@gmu.edu

Daisuke Mashima
Singapore University of Technology and Design
Singapore, Singapore
daisuke_mashima@sutd.edu.sg

Xiaokuan Zhang
George Mason University
Fairfax, VA, USA
xiaokuan@gmu.edu

Abstract

The rapid adoption of Extended Reality (XR) platforms has introduced new user interface (UI) security risks, such as overlapping and invisible object vulnerabilities, stemming from the complex spatial integration of virtual content in users' perceptual environments. Existing detection techniques often require exhaustive, continuous scene scanning, resulting in substantial computational overhead and poor scalability for practical XR applications. In this work, we propose a hybrid detection framework that integrates event-driven object tracking with targeted, periodic analysis to efficiently identify UI integrity violations. Our approach restricts monitoring to objects that enter the user's proximity and field of view, applying focused spatial overlap and invisibility checks through collider and rendering state assessment. Evaluation on commodity XR hardware demonstrates that our framework achieves high detection accuracy with low runtime overhead, providing an effective and scalable defense against critical XR UI security threats.

CCS Concepts

• **Security and privacy → Mobile platform security**; **Software security engineering**.

Keywords

Extended Reality, UI Security, Runtime Detection

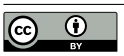
ACM Reference Format:
Bertram Liu, Vamsi Shankar Simhadri, Daisuke Mashima, and Xiaokuan Zhang. 2026. ProViewer: A Proximity- and View-Aware Framework for Detecting UI Security Risks in Extended Reality Applications. In *Second Workshop on Enhancing Security, Privacy, and Trust in Extended Reality (XR Security '26), October 26–30, 2026, Austin, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3842203.3844585

1 Introduction

Augmented Reality (AR) and Virtual Reality (VR), collectively known as Extended Reality (XR), are fundamentally transforming human interaction with digital content. While AR overlays digital information onto the physical environment, VR immerses users in fully virtual spaces. XR technologies are increasingly prevalent across diverse domains such as education [1], healthcare [12], and industrial design [31], marking their emergence as integral components of modern computing platforms. However, by embedding digital objects directly within users' field of view, XR systems introduce novel security challenges that differ substantially from traditional computing environments where interactions are constrained to a flat screen. In XR, adversaries can exploit users' perception and spatial context; for instance, malicious virtual objects may be stealthily positioned or manipulated within three-dimensional space to deceive users, trigger unintended actions, or leak sensitive information. These risks are amplified by the integration of third-party and multi-source content within a shared visual context, complicating the preservation of user interface (UI) integrity and the establishment of clear trust boundaries across mixed digital elements.

Recent work by Cheng et al. [5] systematically analyzes UI security risks in AR, identifying key vulnerabilities such as *overlapping objects*, where multiple virtual objects coexist at the same 3D coordinates with ambiguous rendering or input semantics, and *invisibility*, wherein transparent or non-rendered objects retain the ability to intercept user input or influence scene behavior. These inconsistencies introduce significant security risks for AR UI integrity. To defend against such threats, Liu et al. [9] propose a runtime detection approach: their system, implemented as a Unity plugin, continuously scans scene geometry, collider overlap, and rendering state at a fixed frequency to identify insecure conditions such as overlapping or invisible interactive objects. By highlighting suspicious objects in real time, this defense mechanism enables users and developers to identify and address UI security flaws before they can be exploited.

Although Liu et al. [9] represent an important step forward in XR UI security, their continuous, interval-based scene scanning incurs substantial computational overhead, making the approach impractical for large or complex XR applications. To address this limitation, we introduce ProViewer, a hybrid detection framework that integrates event-driven object tracking with targeted, periodic analysis. Instead of scanning the entire scene, ProViewer activates monitoring only when objects enter both the user's proximity and field of view, significantly reducing computation and enhancing scalability.



This work is licensed under a Creative Commons Attribution 4.0 International License.
XR Security '26, Austin, TX, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2951-5/26/10
https://doi.org/10.1145/3842203.3844585

For overlap detection, ProViewer performs collider-based intersection tests, with fallback to bounds-based checks when colliders are absent or unreliable. For invisibility detection, the system evaluates rendering states as objects enter the user's field of view, flagging inactive renderers as potential UI integrity threats. This hybrid approach allows ProViewer to efficiently and accurately identify key XR UI vulnerabilities while maintaining practical performance at scale. We evaluated ProViewer on a Meta Quest 2 device tethered to a Dell G15 desktop, systematically varying parameters such as scene complexity, view cone angle, and object count. The evaluation results indicate that CPU, GPU, and memory consumption remain stable under different configurations; however, frame rate reduction constitutes the main performance overhead, with more pronounced declines observed under higher scan frequencies and increased scene complexity.

Contributions. The main contributions of this work are:

- We propose ProViewer, a novel hybrid detection framework for XR UI security that integrates event-driven tracking with focused periodic analysis to efficiently and scalably identify UI integrity violations, including overlapping and invisibility vulnerabilities.
- We implement our framework as a Unity plugin, enabling practical deployment in VR settings.
- We conduct a comprehensive empirical evaluation on commodity XR hardware, systematically analyzing the impact of key system parameters on runtime performance.
- We provide a case study on raycasting in scenarios involving invisible or overlapping objects, demonstrating the broad applicability of our approach.

2 Background and Related Work

2.1 UI Security in Traditional Displays

UI security on traditional computing platforms arises from the fundamental limitations of 2D displays, which inherently restrict user's spatial awareness. Adversaries exploit both the visual and input layering of 2D interfaces to manipulate user perception and capture sensitive actions or data. Prominent attacks include clickjacking [8] and UI redressing [16], in which attackers overlay deceptive or transparent elements atop legitimate controls, tricking users into performing unintended actions. Tapjacking [3, 6, 25], prevalent in mobile environments, leverages overlay capabilities to superimpose malicious interface elements, obscuring trusted content and enabling unauthorized input capture or privilege escalation. Furthermore, pixel stealing attacks [17, 23] exploit the rendering pipeline to extract or infer sensitive visual information such as passwords or security tokens by covertly reading screen content from underlying windows or graphical buffers. To mitigate these threats, a variety of defense mechanisms have been deployed. Web platforms employ techniques such as framebusting [27] and content security policies [26] to curtail unauthorized embedding or UI manipulation, reducing the impact of clickjacking and related attacks. Mobile operating systems enforce strict controls on overlay windows [2] and monitor abnormal UI behaviors to detect and alert users of suspicious activity. Collectively, these countermeasures aim to enhance UI integrity and limit the attack surface available to adversaries in 2D environments.

2.2 UI Security Challenges in XR

XR transforms interaction by replacing flat screens with immersive 3D spaces, where actions are mediated by gaze, gesture, and voice. This shift introduces unique security challenges [21, 29], as users must distinguish legitimate UI components from malicious manipulations within a shared spatial context. A central vulnerability arises because multiple virtual elements can occupy overlapping spatial coordinates, letting attackers superimpose malicious objects atop legitimate UI and enable clickjacking-style attacks. XR systems also decouple visibility from interactivity: invisible or transparent objects can still capture input, letting adversaries hijack interactions without visual cues. Integrating third-party or untrusted components further widens the attack surface. Beyond spatial exploits, XR exposes users to perceptual deception attacks [7], including input spoofing, UI forgery, and subtle manipulation of environmental cues that shape user decisions.

2.3 Other Security Issues in XR

Beyond UI manipulation, XR platforms are also vulnerable to a wide range of side-channel attacks. Prior work has demonstrated software-based side channels that exploit shared rendering resources and performance characteristics to infer user interactions, including hand gestures, speech, and keyboard input [4, 13, 28, 30, 33]. Other studies show that users' physical behaviors, such as head movements, can leak private input or 360 degree videos being watched [10, 14, 19]. Hardware-level side channels have also been explored, where external sensing of VR device emissions can reveal user interactions [15]. Additionally, attacks leveraging rendering optimizations and GPU workload variations have been shown to infer sensitive signals such as gaze direction [11, 20]. Beyond side channels, researchers have studied XR firmware security [18] and XR application privacy compliance issues [22, 32].

3 Motivation

XR user interfaces expose fundamentally new security challenges. Recent work by Cheng et al. [5] introduces two key attack classes: (1) *overlapping attacks*, where multiple virtual objects occupy identical spatial coordinates, and (2) *invisibility attacks*, in which interactable objects remain hidden or transparent. These weaknesses degrade user's ability to distinguish authentic UI elements and enable attacks such as clickjacking, denial-of-service, and input forgery. Empirical studies across platforms including ARKit, ARCore, HoloLens, Oculus, and WebXR have demonstrated the real-world impact of these vulnerabilities. Prior work by Liu et al. [9] proposed a runtime detection framework that continuously scans entire XR scenes at fixed rate to identify such UI integrity violations. While effective, this global scanning approach suffers from significant scalability and performance issues as scene size and complexity increase.

Threat Model. We follow the threat model established by prior XR UI security research [5]. We assume a shared-scene XR environment in which a primary application and embedded third-party components can create and manipulate virtual objects within the same scene. A malicious component may place overlapping or invisible objects to interfere with another component's UI elements or the user's interactions.

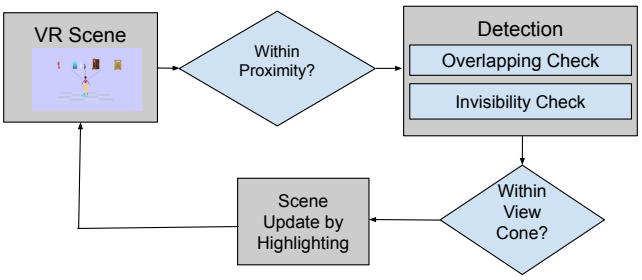


Fig. 1: Workflow

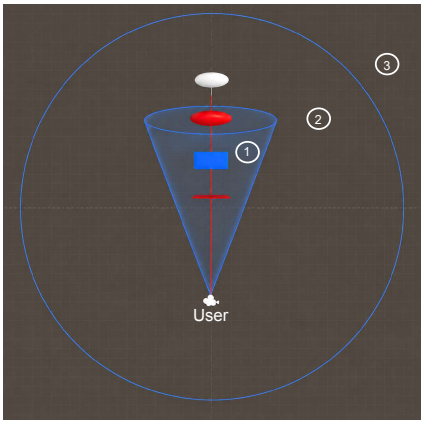


Fig. 2: Top view of the scene showing the view cone from Users POV ① and sphere ②. Objects within ① are highlighted as the object is in users POV, while objects entering ② are actively tracked for detecting overlapping and invisibility.

Goal. Similar to [9], we focus on detecting two primary categories of UI security vulnerabilities from [5]: (1) overlapping attacks, and (2) invisibility attacks. The goal is to identify these UI integrity violations at runtime by monitoring scene-level properties within the user's proximity and view cone.

4 ProViewer Design

To overcome the performance limitations of continuous full-scene scanning, we propose a hybrid detection framework (ProViewer) that combines event-driven tracking with localized periodic analysis. As illustrated in Figure 1, our system restricts security checks to objects that are both within the user's spatial proximity and inside their field of view, thereby minimizing computational overhead. Figure 2 depicts the spatial decomposition: the cone region (①) represents the user's view cone, the sphere region (②) defines the proximity boundary around the user, and the remainder (③) delineates the rest of the scene. The proximity sphere is sized to accommodate natural user head and body movements, ensuring that relevant objects are detected as users look around.

Monitoring is triggered only when objects enter the proximity sphere, at which point they are dynamically registered for targeted analysis. When these objects subsequently enter the user's view cone (①), they are highlighted as security-relevant. This event-driven registration and localized analysis substantially reduces the monitoring scope and enhances runtime efficiency compared to prior global-scanning approaches.

4.1 Proximity-Based Object Identification

The first stage of ProViewer identifies candidate objects most relevant to UI security based on spatial proximity to the user. Instead of exhaustively monitoring all scene objects, we introduce a *proximity sphere*: a spherical region centered at the user's position with a configurable radius r, representing the maximum range of interest. For each scene object o, the Euclidean distance to the user's current position is computed. Objects located beyond radius r are excluded from further analysis. As objects enter the proximity sphere, they are added to an active monitoring set and tracked while they remain within this region. This event-driven mechanism limits resource usage by focusing only on objects likely to be interactable, thereby reducing unnecessary computation. The proximity sphere thus acts as a coarse spatial filter; candidate objects passing this filter undergo further refinement using the field-of-view cone described in Section 4.2, which restricts monitoring to those objects actually within the user's visual field.

4.2 Field-of-View Filtering

After proximity-based filtering, ProViewer further restricts the monitoring set to objects within the user's field of view using a cone-shaped filter aligned with the camera's forward vector. For each candidate object, the system calculates the dot product between the camera's forward direction and the normalized vector from the camera to the object. Objects within less than a preset angular threshold of 70° (corresponding to the view cone) are classified as within the user's visual field and are retained for security analysis. This approach ensures computational resources are dedicated to visually relevant and potentially interactable objects, while excluding those outside the user's direct view.

4.3 Overlapping Detection

ProViewer performs periodic analyses on objects that are within the proximity sphere (②) to detect overlapping UI elements. To address diverse scene structures, we adopt a hybrid detection approach. If objects provide valid collider components, ProViewer executes intersection tests using collider bounds to determine spatial overlap (algorithm 1). If collider information is absent or unreliable, the system resorts to a distance-based method, measuring the Euclidean distance between object centers and comparing it against a small, empirically determined threshold. This two-pronged strategy enables reliable detection of overlap conditions even in environments with incomplete object metadata. Objects identified as occupying the same spatial region are flagged as UI integrity risks; they are visually highlighted if within the view cone (①) to alert users to potential vulnerabilities.

4.4 Invisibility Detection

To detect invisibility attacks, we track the rendering state of objects as they enter the proximity region (②). For each object entering this region, we inspect its renderer component (algorithm 1): if the renderer is disabled while the object is still present in the scene, we classify it as invisible as depicted in Algorithm 1; such objects are visually highlighted if within the view cone (①) to alert users to potential vulnerabilities. Additionally, we periodically re-evaluate the rendering status of all objects in the active set to detect objects

Algorithm 1: Proximity- and View-Aware Detection

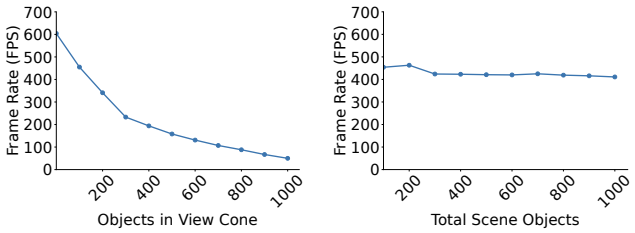
Input: Scene objects O, player camera C, proximity radius r, view angle threshold θ, overlap threshold ϵ
Output: Set of invisible objects I, set of overlapping objects P

$I \leftarrow \emptyset$;
$P \leftarrow \emptyset$;
$R \leftarrow \emptyset$;
$H \leftarrow \emptyset$;
// Step 1: Detect nearby objects and detect invisibility
foreach *object* $o \in O$ **do**
  $d \leftarrow$ Distance($C.position, o.position$);
  if $d \leq r$ **then**
    $R \leftarrow R \cup \{o\}$;
    $ren \leftarrow$ GetRenderer(o);
    // Invisibility Detection
    if $ren = null$ **or** $ren.enabled = false$ **then**
      $I \leftarrow I \cup \{o\}$;
// Step 2: Detect overlapping objects among nearby objects
foreach *pair of objects* $(a, b) \in R$ **do**
  $aCol \leftarrow$ GetCollider(a);
  $bCol \leftarrow$ GetCollider(b);
  if $aCol \neq null$ *and* $bCol \neq null$ **then**
    if *BoundsIntersects*($aCol.bounds, bCol.bounds$) **then**
      $P \leftarrow P \cup \{a, b\}$;
    continue;
  $aBounds \leftarrow$ GetRendererBounds(a);
  $bBounds \leftarrow$ GetRendererBounds(b);
  if $aBounds \neq null$ *and* $bBounds \neq null$ **then**
    if *BoundsIntersects*($aBounds, bBounds$) **then**
      $P \leftarrow P \cup \{a, b\}$;
    continue;
  $dist \leftarrow$ Distance($a.position, b.position$);
  if $dist < \epsilon$ **then**
    $P \leftarrow P \cup \{a, b\}$;
// Step 3: Highlight detected objects only if they are inside the view cone
foreach *object* $o \in I \cup P$ **do**
  $v \leftarrow$ Normalize($o.position - C.position$);
  $\alpha \leftarrow$ Angle($C.forward, v$);
  if $\alpha \leq \theta$ **then**
    $H \leftarrow H \cup \{o\}$;
    HighlightObject(o);
return I, P

that become invisible after initially entering the proximity region. This hybrid strategy that combines event-driven monitoring with periodic reassessment enables the timely detection of both static and transient invisibility attacks during runtime.

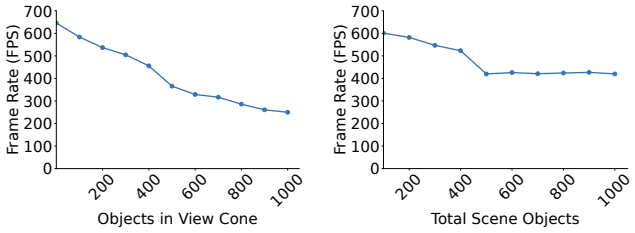
5 Evaluation

We evaluated our detection framework using a series of VR scenes designed to emulate representative VR environments. Experiments were conducted on a Meta Quest 2 connected to a Dell G15 desktop equipped with an AMD Ryzen 5 5600H CPU, 16 GB RAM, and an NVIDIA GeForce RTX 3050 GPU. Evaluation parameters included scanner execution frequency, scene complexity, view-cone angle, maximum detection distance, and the number of objects within the view cone. The scanner continuously analyzed the active field of view to identify invisible and overlapping objects. Key system metrics CPU usage, GPU usage, memory consumption, frame rate, detection delay, and detection accuracy were measured using Unity's integrated profiling tools. Across experiments, CPU, GPU utilization, and memory consumption remained relatively stable, with frame rate being affected by changes in scene complexity, so we mainly evaluate the frame rate. Although the Meta Quest 2 is not the newest VR platform, the detection system is hardware-agnostic, leveraging scene-level features such as visibility, spatial overlap, view direction, and user distance that are standard in Unity-based VR systems. Consequently, the proposed technique is expected to generalize effectively to newer headsets.



(a) Objects in Scene = 1000 **(b) Objects in View Cone = 100**

Fig. 3: Overlap detection performance.



(a) Objects in Scene = 1000 **(b) Objects in View Cone = 100**

Fig. 4: Invisibility detection performance.

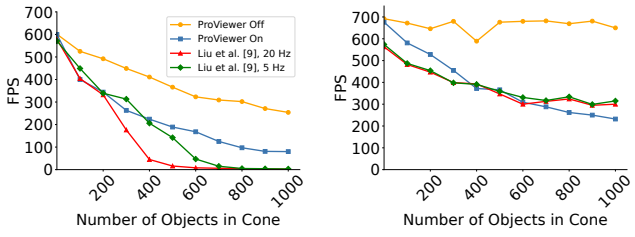
5.1 Overlapping Object detection

Fixed scene size, varying objects in cone. To assess the effect of objects within the user's view, we fixed the total scene size at 1000 objects and varied the number of objects inside the cone from 0 to 1000 (Figure 3a). The results demonstrate that the detector's performance is primarily determined by the number of objects inside the cone: as this number increases, FPS drops from 454 (100 objects in cone) to 50 (1000 objects), an 89% reduction. This trend confirms that per-frame computational cost scales with the number of actively monitored objects within the cone.

Fixed objects in cone, varying total scene size. We next analyzed scalability with respect to overall scene complexity by holding the number of objects inside the cone constant at 100, while varying the total number of objects in the scene from 100 to 1000. Across this 10-fold increase in scene size, FPS remains relatively stable, decreasing only from 454 to 411 (9.5% reduction). This result indicates that detector runtime is largely insensitive to total scene size, and is instead dominated by the number of objects present within the cone. Figure 3b illustrates this relationship.

5.2 Invisibility

Fixed scene size, varying objects in cone. We evaluated invisibility detection performance by fixing the total number of objects in



(a) Overlapping Detection **(b) Invisibility Detection**

Fig. 5: Performance comparison (ProViewer vs. Liu et al. [9]). ProViewer outperforms Liu et al. [9] on overlapping detection and achieves comparable invisibility detection performance, with a slight drop in dense scenes.

the scene at 1000 and varying the number of objects within the view cone from 0 to 1000 (Figure 4a). The results show that frame rate (FPS) is primarily determined by the number of objects inside the cone: as this number increases, FPS drops from 584 (100 objects in cone) to 250 (1000 objects), corresponding to a 57.2% reduction. This trend indicates that the computational cost of the detector scales with the number of monitored objects within the cone, rather than overall scene size.

Fixed objects in cone, varying total scene size. To assess the impact of scene complexity, we fixed the number of objects in the cone at 100 and increased the total number of scene objects from 100 to 1000 (Figure 4b). Across this 10-fold increase, FPS remained relatively stable, decreasing from 564 to 420 (a 25.5% reduction). These results demonstrate that the detector's runtime is largely insensitive to total scene size, and is instead dominated by the number of objects actively processed within the cone.

5.3 Comparison with Prior Work [9]

Overlapping. We evaluated overlapping detection by varying the number of objects from 0 to 1000, with all objects placed inside the view cone. ProViewer introduces moderate overhead at low object counts: FPS drops from 525 to 400 at 100 objects and from 492 to 344 at 200 objects. The overhead increases in denser scenes, with FPS dropping from 366 to 189 at 500 objects and from 254 to 80 at 1000 objects. Figure 5a compares ProViewer with Liu et al. [9] in terms of overlapping detection. Their scanner drops sharply as object count increases, reaching 16 FPS at 500 objects and 2 FPS at 1000 objects for 20 Hz scanning, and 142 FPS and 3 FPS for 5 Hz scanning. In contrast, ProViewer maintains 189 FPS at 500 objects and 80 FPS at 1000 objects. This shows that ProViewer scales better in dense scenes by limiting detection to objects in the user's active view cone instead of repeatedly scanning the full scene.

Invisibility. We evaluated invisibility detection by varying the number of objects from 0 to 1000, with all objects placed inside the view cone. ProViewer introduces limited overhead at low object counts, reducing FPS from 672 to 581 at 100 objects and from 646 to 528 at 200 objects. As scene density increases, the overhead becomes more noticeable. At 500 objects, FPS decreases from 676 to 366, and at 1000 objects, from 650 to 232. Figure 5b compares ProViewer with Liu et al. [9] in terms of invisibility detection. The prior approach achieves 347 FPS (20 Hz) and 359 FPS (5 Hz) at 500 objects, which is comparable to ProViewer's 366 FPS. At 1000 objects, the prior approach achieves 300 FPS and 315 FPS, while ProViewer achieves 232 FPS. Unlike overlapping detection, in invisibility detection, while ProViewer performs good when the number of objects is small, ProViewer performs worse than Liu et al. [9] when the number of objects is larger than 600. We conjecture that the performance degradation is due to additional sphere and view cone checks for each object, which offsets the performance gain. Although invisibility detection introduces slight overhead, ProViewer maintains sufficient performance for real-time use as the FPS is still higher than 120 [24].

6 Case Study: Raycasting Attack Detection

Raycasting is the primary interaction paradigm in XR systems for determining which virtual object a user is targeting. In this mechanism, a ray is projected from the user's input source—such as a controller, gesture, gaze, or screen touch—into the scene; the first intersected object traditionally receives the interaction event. However, both overlapping and invisibility attacks undermine this pipeline. Adversaries may place overlapping objects at the same coordinates to hijack user input, or insert invisible objects along the ray path to surreptitiously intercept interactions. Standard single-hit raycasting (e.g., `Physics.Raycast`) exclusively detects the closest intersecting object. This design overlooks further objects along the ray, missing hidden or overlapped objects and thereby masking the presence of UI security violations.

To systematically expose these vulnerabilities, ProViewer uses multi-hit raycasting via `Physics.RaycastAll`. For each user interaction ray, our system retrieves and orders all intersected objects by distance from the user. This comprehensive interaction chain enables the explicit detection of all objects along the intended input path, including non-visible or spatially overlapping elements. Analyzing the entire set of intersected objects allows ProViewer to effectively identify instances of the two attack classes: if invisible or overlapping objects are returned in the ray path, the system highlights these anomalies to the user. In our evaluation, ProViewer achieves 100% accuracy in detecting invisible and overlapping objects encountered through raycasting, ensuring user awareness of potential UI integrity violations during interaction.

7 Limitations and Future Work

While our hybrid detection framework efficiently identifies overlapping and invisibility-related UI security risks with substantially lower overhead than continuous scene scanning, several limitations remain. First, our proximity and view-cone-based filtering achieves performance gains by limiting analysis to objects relevant to the user's immediate interaction space. However, this selective monitoring may introduce latency, as objects located outside the active region are only analyzed once they enter the user's proximity or field of view. Second, our evaluation is currently limited to controlled, Unity-based VR environments on a Meta Quest 2 device tethered to a desktop system. Extending the evaluation to additional XR platforms, engine implementations, and conducting studies with real-world users are important directions for future work. Third, our framework has thus far been evaluated only in synthetic environments using artificial objects. It has not yet been assessed in real-world XR scenes with real-world objects. Future

work should include comprehensive evaluation in realistic scenarios to validate the framework's effectiveness and generalizability. Fourth, our current framework focuses on detecting scene-level UI security risks, mainly overlapping and invisible objects. Fifth, our raycasting case study is rudimentary; a malicious component may generate synthetic raycasts or manipulate interaction targets in ways that are not fully captured by object-level scene analysis. Detecting such cases would require input-level monitoring, which we leave as future work. Lastly, benign applications may legitimately use invisible/overlapping objects as part of their intended functionality. Future work could include a developer-defined whitelist to distinguish such use-cases from potentially malicious scenarios, thereby reducing false positives.

8 Conclusion

In this paper, we introduced ProViewer, a hybrid framework for detecting XR UI security vulnerabilities related to overlapping and invisible objects. By focusing analysis on objects within the user's proximity and field of view, ProViewer reduces the inefficiency of continuous full-scene scanning while maintaining effective detection of security-critical UI behaviors. Our evaluation on Meta Quest 2 demonstrates that system performance is primarily determined by the number of objects within the active monitoring region, rather than the total scene complexity. These results indicate that localized, event-driven runtime detection offers a scalable and practical approach to strengthening XR UI security.

Acknowledgement

Bertram Liu was supported in part by a CCI NoVa Node Cybersecurity Undergraduate Research Assistantship. This research was funded in part by 4-VA, a collaborative partnership for advancing the Commonwealth of Virginia.

References

[1] Abdullah M Al-Ansi, Mohammed Jaboob, Askar Garad, and Ahmed Al-Ansi. 2023. Analyzing augmented reality (AR) and virtual reality (VR) recent development in education. *Social Sciences & Humanities Open* 8, 1 (2023), 100532.

[2] Android Developers. 2024. Behavior Changes: Apps Targeting Android 15 or Higher. https://developer.android.com/about/versions/15/behavior-changes-15. Accessed: 2026-06-16.

[3] Philipp Beer, Marco Squarcina, Sebastian Roth, and Martina Lindorfer. 2025. {TapTrap}:{Animation-Driven} Tapjacking on Android. In *34th USENIX Security Symposium*. 3317–3335.

[4] Derin Cayir, Reham Mohamed Aburas, Riccardo Lazzeretti, Marco Angelini, Abbas Acar, Mauro Conti, Z Berkay Celik, A Selcuk Uluagac, et al. 2025. Speak Up, I'm Listening: Extracting Speech from Zero-Permission VR Sensors.. In *NDSS*.

[5] Kaiming Cheng, Arkaprabha Bhattacharya, Michelle Lin, Jaewook Lee, Aroosh Kumar, Jeffery F Tian, Tadayoshi Kohno, and Franziska Roesner. 2024. When the user is inside the user interface: An empirical study of {UI} security properties in augmented reality. In *33rd USENIX Security Symposium*. 2707–2723.

[6] Yanick Fratantonio, Chenxiong Qian, Simon P Chung, and Wenke Lee. 2017. Cloak and dagger: from two permissions to complete control of the UI feedback loop. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1041–1057.

[7] Hilda Hadan, Lydia Choong, Leah Zhang-Kennedy, and Lennart E Nacke. 2024. Deceived by immersion: A systematic analysis of deceptive design in extended reality. *Comput. Surveys* 56, 10 (2024), 1–25.

[8] Lin-Shung Huang, Alex Moshchuk, Helen J Wang, Stuart Schecter, and Collin Jackson. 2012. Clickjacking: Attacks and defenses. In *21st USENIX Security Symposium*. 413–428.

[9] Bertram Liu, Vamsi Shankar Simhadri, and Xiaokuan Zhang. 2025. Hunting Insecure UI Properties in Extended Reality. In *Proceedings of the Twenty-sixth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*. 462–467.

[10] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. 2022. Holologger: Keystroke inference on mixed reality head mounted displays. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 445–454.

[11] Paul Maynard, Harris Amjad, Camila Molinares, Bo Ji, and Brendan David-John. 2026. EyeSpy: Inferring Eye Gaze via Side-Channel Attacks Against Foveated Rendering. *arXiv preprint arXiv:2605.27939* (2026).

[12] Leslie Mertz. 2019. Virtual reality pioneer tom furness on the past, present, and future of VR in health care. *IEEE pulse* 10, 3 (2019), 9–11.

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

[14] Anh Nguyen, Xiaokuan Zhang, and Zhisheng Yan. 2024. Penetration Vision through Virtual Reality Headsets: Identifying 360-degree Videos from Head Movements. In *USENIX Security'24*.

[15] Tao Ni, Yuefeng Du, Qingchuan Zhao, and Cong Wang. 2024. Non-intrusive and unconstrained keystroke inference in vr platforms via infrared side channel. *arXiv preprint arXiv:2412.14815* (2024).

[16] Marcus Niemietz and Jörg Schwenk. 2012. Ui redressing attacks on android devices. *Black Hat Abu Dhabi* (2012), 71–81.

[17] Sioli O'Connell, Lishay Aben Sour, Ron Magen, Daniel Genkin, Yossi Oren, Hovav Shacham, and Yuval Yarom. 2024. Pixel thief: Exploiting {svg} filter leakage in firefox and chrome. In *33rd USENIX Security Symposium*. 3331–3348.

[18] Vamsi Shankar Simhadri, Yichang Xiong, Habiba Farrukh, and Xiaokuan Zhang. 2025. Virtual Reality, Real Problems: A Longitudinal Security Analysis of VR Firmware. In *Proceedings of the 2025 ACM SIGSAC conference on computer and communications security*. ACM.

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

[20] Seonghun Son, Chandrika Mukherjee, Reham Mohamed Aburas, Berk Gulmezoglu, and Z Berkay Celik. 2025. Side-channel Inference of User Activities in AR/VR Using GPU Profiling. *arXiv preprint arXiv:2509.10703* (2025).

[21] Ali Teymourian, Andrew M Webb, Taha Gharaibeh, Arushi Ghildiyal, and Ibrahim Baggili. 2025. {SoK}: Come Together–Unifying Security, Information Theory, and Cognition for a Mixed Reality Deception Attack Ontology & Analysis Framework. In *34th USENIX Security Symposium*. 1475–1492.

[22] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. 2022. {OVRseen}: Auditing network traffic and privacy policies in oculus {VR}. In *31st USENIX security symposium*. 3789–3806.

[23] Alan Wang, Pranav Gopalkrishnan, Yingchen Wang, Christopher W Fletcher, Hovav Shacham, David Kohlbrenner, and Riccardo Paccagnella. 2025. Pixnapping: Bringing pixel stealing out of the stone age. In *Proceedings of the 2025 ACM SIGSAC CCS*. 3266–3280.

[24] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. 2023. Effect of frame rate on user experience, performance, and simulator sickness in virtual reality. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[25] Xianbo Wang, Shangcheng Shi, Yikang Chen, and Wing Cheong Lau. 2022. PHYjacking: Physical Input Hijacking for Zero-Permission Authorization Attacks on Android.. In *NDSS*.

[26] Wikipedia contributors. 2026. Content Security Policy — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Content_Security_Policy&oldid=1347712092 [Online; accessed 16-June-2026].

[27] Wikipedia contributors. 2026. Framekiller — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Framekiller&oldid=1346273249 [Online; accessed 16-June-2026].

[28] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. 2023. 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)*. IEEE, 3382–3398.

[29] Yanming Xiu and Maria Gorlatova. 2025. Detecting visual information manipulation attacks in augmented reality: a multimodal semantic reasoning approach. *IEEE Transactions on Visualization and Computer Graphics* (2025).

[30] Zhengkun Ye, Ahmed Tanvir Mahdad, Yan Wang, Cong Shi, Yingying Chen, and Nitesh Saxena. 2025. Bpsniff: Continuously surveilling private blood pressure information in the metaverse via unrestricted inbuilt motion sensors. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4356–4374.

[31] Xue Yu, Stephen DiVerdi, Akshay Sharma, and Yotam Gingold. 2021. Scaffoldsketch: Accurate industrial design drawing in vr. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 372–384.

[32] Yuxia Zhan, Yan Meng, Lu Zhou, Yichang Xiong, Xiaokuan Zhang, Lichuan Ma, Guoxing Chen, Qingqi Pei, and Haojin Zhu. 2024. VPVet: Vetting Privacy Policies of Virtual Reality Apps. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

[33] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. 2023. It's all in your head (set): Side-channel attacks on {AR/VR} systems. In *32nd USENIX Security Symposium*. 3979–3996.